

UNIVERSITE CATHOLIQUE DE LOUVAIN

FACULTE DES SCIENCES APPLIQUEES

DÉPARTEMENT D'INGÉNIERIE INFORMATIQUE

**Development of multimodal user interfaces by  
interpretation and by compiled components :  
a comparative analysis between InterpiXML and  
OpenInterface.**

Promoteur : Jean Vanderdonckt

Mémoire présenté en vue  
de l'obtention du grade  
de Licencié  
en Informatique

par :

- Goffette Yann

- Louvigny Henri

Louvain-la-Neuve  
Année académique 2006-2007



UNIVERSITE CATHOLIQUE DE LOUVAIN

FACULTE DES SCIENCES APPLIQUEES

DÉPARTEMENT D'INGÉNIERIE INFORMATIQUE

**Development of multimodal user interfaces by  
interpretation and by compiled components :  
a comparative analysis between InterpiXML and  
OpenInterface.**

Promoteur : Jean Vanderdonckt

Mémoire présenté en vue  
de l'obtention du grade  
de Licencié  
en Informatique

par :

- Goffette Yann

- Louvigny Henri

Louvain-la-Neuve  
Année académique 2006-2007

## Acknowledgements

We first would like to thank, Mister Vanderdonckt, our supervisor, for his support and his help during all the development of this thesis : it was absolutely invaluable. We would also thanks him to give us the opportunity to participate at the eNTERFACE 2007 workshop. This workshop was a particularly great and rich experience both for ourself and for the fulfilment of this thesis.

About this workshop we thank Lionel Lawson and Marcos Serrano for their help on the Open Interface platform.

Obviously we thank all volunteers who participated to our tests.

Finally We would like to thank our families and friends for their support and encouragement during the year it took us to realize this work and write this thesis.

Henri would like to give a special thanks to his parents André and Josiane to his family Emilie and Anne-Catherine, his flatmates Valérie, Tristan and Vincent and all his friends.

Yann would like to give a special thanks to his parents, Marie, and his flatmates and friends : Bossic, Bernard and Cyril.

# Table of contents

1. Introduction.....	12
1.1 Context.....	12
1.1.1 Man-machine interfaces.....	13
1.1.2 Multimodality.....	13
Advantages of multimodality.....	15
Multimodal Systems are more robust.....	15
Multi-modal communication is often more simple.....	15
Multimodal Systems are more flexible.....	16
1.1.3 The CARE properties.....	16
Equivalence.....	17
Assignment.....	17
Redundancy.....	18
Complementarity.....	18
1.1.4 CARE-Like properties of the user.....	18
1.1.5 CARE properties and CARE-like properties : Conclusion.....	19
1.1.6 Gesture.....	19
Pen-based gesture.....	19
Hand-based gesture.....	20
1.1.7 Gesture recognition.....	20
1.2 Motivations.....	21
1.2.1 Potential commercial issue.....	21
1.3 Goals.....	22
1.4 Reading plan.....	23
2. State of the art.....	24
2.1 The current existing gestures.....	24
2.1.1 Pen Based Gesture.....	24
All in one gesture plug in for Firefox.....	24
Matis system.....	25
Handwriting recognition.....	26
Operating Systems.....	26
2.1.2 Hand Gestures.....	26
2.2 Gestures Qualities.....	28
2.2.1 Pen based gestures qualities.....	28
Iconicity.....	28
Learnability.....	29
Gesture recognizer recognizability.....	29
Compatibility and coherence.....	29
2.2.2 Hand based gestures qualities.....	30
Iconicity.....	30
Learnability.....	30
Gesture recognizer recognizability.....	31
Compatibility and coherence.....	31
2.3 Actions set on interfaces.....	31
Windows managing actions.....	31
Browsability actions.....	32
Validations actions.....	32
Characters and numbers .....	32
2.4 Existing Toolkits.....	32
2.4.1 Pen-based Toolkits.....	32

PenBuilder.....	32
Microsoft XP tablet Edition development kit.....	33
2.4.2 Hand-based Toolkits.....	33
Isight Sony.....	33
Georgia Tech Gesture Toolkit (GT2K).....	34
2.5 Quill Toolkit.....	34
Introduction.....	34
General principle.....	34
Rubine's algorithm.....	35
Advantages of Quill.....	36
Satin.....	38
2.6 HandVu Toolkit.....	39
Camera.....	39
Hand detection.....	39
Hand tracking.....	40
Posture recognition.....	40
2.7 Specification language choice.....	42
2.7.1 UsiXML.....	42
The language.....	42
The interpreter : InterpiXML.....	44
2.8 OpenInterface.....	45
Introduction.....	45
OpenInterface architecture.....	46
General Principle.....	47
a) The component CIDL description.....	47
b) The pipeline PDCL description.....	48
Similar project.....	48
eNTERFACE workshop.....	49
3. Design of multimodal interfaces.....	51
3.1 Multimodal architectures.....	51
a) MVC architecture.....	52
b) ARCH architecture.....	54
3.2 Pen-based gestures.....	56
Windows managing actions.....	56
Browsability actions.....	57
Validation actions.....	58
Characters gestures.....	58
3.3 Hand gestures .....	60
Windows managing actions.....	60
Validations actions.....	62
4. InterpiXML Development.....	63
From InterpiXML v1.0 to v1.1.....	63
Adaptation to UsiXML v1.8.0.....	64
Adaptation to multimodality.....	64
Architecture.....	67
4.1 InterpiXML and hand-based recognition.....	72
4.1.1 Architecture.....	72
4.1.2 Implementation.....	72
4.1.3 Examples.....	74
4.1.4 Evaluation.....	74

4.2 InterpiXML and pen-based recognition.....	75
4.2.1 Architecture.....	75
4.2.2 Implementation.....	75
4.2.3 Examples.....	77
4.2.4 Evaluation.....	77
4.3.1 Examples.....	78
4.3.2 Evaluation.....	78
4.4 General evaluation.....	79
5. OpenInterface integration.....	80
5.1 OpenInterface and hand-based recognition.....	83
5.1.2 Implementation.....	84
5.1.3 Examples.....	84
5.1.4 Evaluation.....	85
5.2 OpenInterface and pen-based recognition.....	85
5.2.1 Architecture.....	86
5.2.2 Implementation.....	86
5.2.3 Examples.....	87
5.2.4 Evaluation.....	87
5.3 OpenInterface with hand and pen-based recognition.....	88
5.3.1 Architecture.....	88
5.3.4 Evaluation.....	89
5.4 General evaluation.....	89
5.5 InterpiXML integration to OpenInterface.....	90
5.5.1 Architecture.....	90
5.5.2 Implementation.....	91
5.5.3 Example.....	91
5.5.4 Evaluation.....	92
6. Tests.....	93
The goals.....	93
The experiment itself.....	93
The experiment protocol.....	95
The experiment conditions.....	96
The evaluation forms.....	96
The pre-test.....	97
The participants.....	97
Hypothesis.....	97
The Results and conclusions.....	97
1) Volunteers don't see the differences between OpenInterface and InterpiXML.....	97
2) Volunteers should prefer tablet to webcam.....	99
3) Webcam accuracy may disappoint users.....	100
4) Volunteers don't use the 2 modalities simultaneously.....	100
5) Experimented users of tablet should accomplish task more rapidly.....	100
6) Learning is quiet important for both modalities.....	101
Other conclusions and interesting results.....	103
Conclusions.....	104
Bibliography.....	106
Books, periodics et papers.....	106
Web links.....	108

Appendix..... 109

Demographic forms..... 110

Evaluation forms..... 111

CD Content..... 114



## Figures index

Figure 1: The Put That There system : Bolt 1980.....	14
Figure 2: Pen-based gesture.....	19
Figure 3: Hand-gesture.....	19
Figure 4: Wii platform.....	20
Figure 5: Wii wheel controleur.....	20
Figure 6: Gartner Hype Cycle 2006 [Gar 06].....	21
Figure 7: Gartner Hype Cycle 2006 [Gar 06].....	22
Figure 8: Set of possible actions with gesture mouse with All in one gesture plug in for Firefox ..	25
Figure 9: The MATIS application.....	25
Figure 10: Graffiti alphabet.....	26
Figure 11: Hand Gestures illustration.....	27
Figure 12: Hand gesture recognition system for replacing a mouse.....	28
Figure 13: Delete and Copy iconic gestures.....	29
Figure 14: u and v different design.....	29
Figure 15: Example of gesture coherence.....	30
Figure 16: Thumb up for Ok.....	30
Figure 17: One information to learn to do four actions (Up, Down, Left, Right).....	31
Figure 18: Quill illustration : .....	35
Figure 19: Aspect feature.....	37
Figure 20: Curviness feature.....	37
Figure 21: Rondaboutness feature.....	37
Figure 22: Density feature.....	38
Figure 23: Posture for hand detection.....	39
Figure 24: Recognized postures.....	41
Figure 25: The Cameleon reference framework for multi-target UIs.....	42
Figure 26: Explorer InterpiXML v1.0.....	44
Figure 27: OpenInterface Logo.....	45
Figure 28: OpenInterface currents components.....	45
Figure 29: OpenInterface architecture.....	46
Figure 30: Similar logo.....	48
Figure 31: Similar context.....	49
Figure 32: Bogaziçi university.....	50
Figure 33: Typical information processing flow in a multimodal architecture designed for speech and gesture.....	52
Figure 34: MVC architecture.....	53
Figure 35: The ARCH model.....	54
Figure 36: ARCH architecture for dialogue oriented systems.....	55
Figure 37: Windows managing actions pen gestures.....	55
Figure 38: Browsability actions pen gestures.....	56
Figure 39: Validation actions pen gestures.....	57
Figure 40: Characters pen gestures.....	58
Figure 41: Numbers pen gestures.....	59
Figure 42: Windows managing hand gestures.....	59
Figure 43: Browsability actions hand gestures.....	60
Figure 44: Validation action hand gestures.....	61
Figure 45: Reading plan for implementation chapters.....	62
Figure 46: First step in the modality – interfaces communication.....	64
Figure 47: Second step in the modality – interfaces communication.....	64
Figure 48: Third step in the modality – interfaces communication.....	65
Figure 49: InterpiXML ARCH architecture.....	66

Figure 50: InterpiXML architecture.....	67
Figure 51: Package be.ac.ucl.isys.InterpiXML.multiModale.....	68
Figure 52: handVu package.....	69
Figure 53: quill package.....	69
Figure 54: Typical execution between HandVu software and interfaces produce by InterpiXML...	70
Figure 55: Gesture name - message posted - action associations.....	72
Figure 56: InterpiXML with hand recognition modality.....	73
Figure 57: Gesture name – Message posted – action associations.....	75
Figure 58: Screenshot pen-based modality and InterpiXML.....	76
Figure 59: The QuillModality Keyboard.....	76
Figure 60: Both modality connected to InterpiXML.....	77
Figure 61: Mapping Component.....	80
Figure 62: HandVu component pipe in OpenInterface.....	83
Figure 63: HandVu and a GUI with OpenInterface print screen.....	84
Figure 64: Quill-ImageViewer pipeline.....	85
Figure 65: Pen based Recognition on OpenInterface.....	86
Figure 66: Pipeline pen-based and hand recognition on OpenInterface.....	87
Figure 67: Pen-based and hand recognition on OpenInterface.....	88
Figure 68: Pipe for InterpiXML and both modalities.....	89
Figure 69: InterpiXML - OpenInterface connection with 2 modalities.....	91
Figure 70: Task model for test interface.....	93
Figure 71: French fries order form .....	93
Figure 72: The learning Interface.....	94
Figure 73: The experiment room.....	95
Figure 74: InterpiXML and OpenInterface Users comparaison.....	97
Figure 75: Appreciation.....	98
Figure 76: Modality used when choice offered.....	99
Figure 77: Time to accomplish the task.....	101
Figure 78: Task achieving.....	101

## Code index

Code 1: Interface reaction to event implementation.....	72
Code 2: Code For the command and charcter recognizers.....	75
Code 3: CIDL factory code for the Mapping Component factory.....	80
Code 4: CIDL Sink code for the Mapping Component.....	81
Code 5: CIDL Source code for the Mapping Component.....	81
Code 6: HandVuMapping.txt : Translation file for HandVu.....	81

# 1. Introduction

The last ten years, the man machine interface have known a very important evolution. In the 50's, it was necessary to resort to plug boards, on which one plugged in cables connecting two operators to program mathematical operations on the electromechanical tabulators, remote ancestors of our programmable calculators. In the 60's, systems became able to interpret a line of order : the keyboard was essential, accompanied soon by the screen. As this time, more convivial modes of interaction with the machines were searched in particular at the Xerox Palo Alto Research Center. In 1964, Douglas C. Engelbart had conceived the principles of the modern graphic interface : instead of posting lines of orders the ones following the others, the screen could accomodate windows in which menus were posted, which one could reach by moving a pointer with a two metal wheels mouse.

Screen, keyboard, mouse : the three fundamental elements of the interface of our computers were joined together. In 1979 after the visit at the Alto Research Center of a young man called Steve Jobs, the invention had spread for personal use. The young employer of Apple was going to equip the first Macintosh, launched in 1984, of a graphic interface and a mouse.

The interface man machine hardly moved since, at least for the private individual. But today things seems to get in move, the multimodality is now getting present. What it is and how does it works, that's our challenge to explain you in this thesis.

## 1.1 Context

The context of this thesis is then the man-machine interfaces, especially the study and the implementation of the multimodal interfaces. Nowadays this recent field of studies contribute a lot in the man-machine interfaces study. But before starting out, let's define some important concepts.

### 1.1.1 Man-machine interfaces

*« Il serait sot de nier l'importance de la communication efficace entre l'homme et la machine, aussi bien que l'inverse. Ma prévision est toutefois que la vraie révolution des prochaines décennies viendra davantage encore de ce que les hommes ont à se dire par l'intermédiaire des machines » : James Cannavino*

#### **Definition :**

The quotation of the strategic director of IBM gives us a idea of in what consist in the man-machines interfaces. But we should give more precises définitions.

*« It means the aggregate of interaction human-machine, man-machine interface (MMI) studies the ways humans interacts with computers or between themselves with the help of the computers, but also the way to develop computer-systems which are ergonomic, it means effective, and easy to use or more generally, adapted to their context of use.»*

So if we had to summarize, we would say that it consist in a set of device and softwares allowing a user to communicate easily with the computer. So it consists in :

- A mean of communication between humans and machine in general (a modality).
- A field of study having the objective to make this communication transparent, natural, efficient and effective.

### 1.1.2 Multimodality

What does means multimodal ? The Etymology of this word, informs us that the prefix multi comes from the Latin *multos* : many and the suffix *modal* is the adjective of the word “mode” which mean the particular way an action is done. So the definition of a multimodal interface could be :

*« Interface which propose to his users, an numerous different interactions mode.»* but also :

Multimodal interfaces process two or more combined user input modes— such as speech, pen, touch, manual gestures, gaze, and head and body movements— in a coordinated manner with multimedia system output. They are a new class of interfaces that aim to recognize naturally occurring forms of human language and behaviour, and which incorporate one or more recognition-based technologies (e.g., speech, pen, vision).[OVIA 02]



***Figure 1: The Put That There system :  
Bolt 1980***

In fact, we don't have to confuse mode and modality. The mode is an abstract way to interact with the computer by using one of the sense of human body while a modality is the realisation of the communication mode. For example, we use the modality speech recognition for the vocal mode. We can also have more than one modality for a mode for example touch mode can achieve the mouse and a pen-based gesture recognition.

The first real multimodal system was the «put that there» system which combined the manipulation of graphical object and the speech recognition [BOLT 80]. This «put that there» system, in figure 1, was then combining two interaction mode. An interaction modality is a manner of communicating with the machine and a way for the machine to communicate with the user. Screen, keyboard and mouse are the most famous interaction modality. Nowadays, an huge number of modalities are flourishing : cameras, varying minces, touchable screens, pen tablets, speech recognition,... In short, users have the choice. So the user should be able to choose the way he wants to interact with their computers. Nonetheless, the large part of applications only deals with two inputs modalities which are mainly the mouse and the keyboard.

In fact, if software engineers set the speech recognition as the main modality, users will be probably disappointed by the accuracy of the speech recognizer if the environment is noisy. That's where multimodality becomes interesting. Instead of having only the speech recognition as input we could have also a lips reader input. The user has now the choice between speaking or lips reading or even both. Then in a noisy place, lips reading will provide another input to speech recognition that will help the speech recognition in his work. This is one of the advantage provided by the multimodality. The next section is covering all the advantages of the multimodality.

Humans also are communicating multimodally. In fact, to increase the listener's understanding the speaker often use his hands or expressions on his face or even body gestures. It's then obvious to provide to computer's users a more natural way to communicate with computer. Providing either a more intuitive way of communication for the user and increase the computer's understanding of the users queries.

## **Advantages of multimodality**

### ***Multimodal Systems are more robust:***

As we saw in the introduction, lips reading combined with speech recognition provide another input stream that can support the speech recognition in the noisy environment. If the speech recognition is hesitating between the word «tough» and «thought», as the figure of the mouth is completely different the system will achieve the right command. Ambiguities are resolved due to multimodality. In this case, we are talking about «mutual disambiguity» [OVIA 02]. This shows that multimodal systems are more robust than unimodal systems. The weaknesses of an interaction mode ( depending on the current environment or not) are overcome by the strengths of the other modalities [OVIA 02]. But those systems are not robust only due to mutual disambiguity but also due to users. First it's because users will select the input mode that they judge less error-prone. Secondly, users language is more simple (see next advantages of multimodality). And finally, users tend to switch from interaction mode after system recognition errors. This facilitate error recovery [OVIA 02].

In two recent studies involving 4600 multimodal commands, a multimodal it has been found that mutual disambiguity and error suppression was about from 19 to 41 %. (compared to unimodal systems) [OVIA 02]. Mutual disambiguation involves disambiguation of signal or semantic-level information in one error-prone input mode from partial information supplied by another. Mutual disambiguation can occur in a multimodal architecture with two or more semantically rich recognition-based input modes. It leads to recovery from unimodal recognition errors within a multi-modal architecture, with the net effect of suppressing errors experienced by the user [OVIA 02].

### ***Multi-modal communication is often more simple:***

As we said in the previous section the communication provided by the multimodal systems is more natural and intuitive. In fact, some commands are easily expressed multimodally. For example, when interacting with spacial stuffs such as graphical objects users do prefer say «move this here» and point the object to move with a pointing device instead of saying «put the red cross next to highest red building» [OVIA 02]. However using a pointing device add some cognitive workload to the user [RUGE 03]. This example is showing an important characteristic of an interface, the accessibility. The accessibility is defined by the easiness with which the users can use the functions of an interface independently of their constraints. A user suffering from blindness could fill the a form with vocal recognition as input and sound as output instead of using keyboard and not see the results on the screen.

### ***Multimodal Systems are more flexible:***

Multimodal systems are more flexible according to the fact that user are choosing their interaction mode. We can here introduce the notion of utilisability. The utilisability characterize the easiness to use the interface. For example, if we use a palm an we wish to fill a form on a web page it will be easier to fill the different widgets with speech recognition instead of drawing the different signs representing the different characters. In contrast with an unimodal system where user has only to deal with pen.

*«With eight tentacles and the ability to shift colours rapidly, the intelligent octopus is a master at learning, adapting to, and controlling its environment. To improve their coverage, reliability and usability, multi-modal interfaces likewise are being designed that can automatically learn and adapt to important user, task, and environmental parameters.» [OVIA 04]*

The goal of the multimodality is then to extend their utility to more challenging mobile environment and larger group of users. The multimodality can then adapt to his environment, in a noisy place, instead of a speech recognition we can use a mouth-listening recognition as input and as output, a sound by bright weather and a light in dark ones. But also will enable us to provide more robust and flexible systems.

### **1.1.3 The CARE properties**

The care properties have been designed to evaluate and characterize the aspect of any multi-modal interaction that may occurs between the interaction techniques available in a multi-modal user interface. Those properties are : The complementarity, the assignment, the redundancy and the equivalence. CARE properties have been designed by Amodeus European project in 1995 but are mainly the work of Laurence Nigay [NIGA 95a].

Multi-modal user interfaces support interaction techniques which may be used sequentially or concurrently and independently or combined synergically [NIGA 95a]. That's why current evaluation techniques such as consistency, observability and pre-emptiveness are not sufficient.

The formal expression of the CARE properties relies on the notions of state, goal, modality, and temporal relationships. The explanation of the CARE properties are taken from [NIGA 95a].

A *state* is a vector of observables, that is, a set of properties that can be measured at a particular time to characterise a situation. A *goal* is a state that an agent intends to reach. An *agent*, e.g., a user, or the system, or a component of the system, is an entity capable of initiating the performance of actions. A *modality* is an interaction method that an agent can use to reach a goal. To model the expressive power of a modality  $m$ , that is, its capacity to allow an agent to reach state  $s'$  from state  $s$  in one step, we use the function  $Reach(s, !m, !s')$ . A sequence of successive steps (or states) is called an *interaction trajectory*. This generic definition of a modality can be interpreted at different levels of refinement. For example, a modality could be specified in general terms as 'using speech', or more specifically as 'using a microphone'. Both of these interpretations are valid.



A *temporal relationship* characterises the use over time of a set of modalities. The use of these modalities may occur simultaneously or in sequence within a *temporal window*, that is, a time interval. Alternatively, only one modality from a set may be used. Let  $Pick(s, !m, !s')$  be a predicate that expresses the use of  $m$  among a set of modalities to reach  $s'$  from  $s$ . Modalities of a set  $M$  are used simultaneously (or in parallel) if, within a temporal window, they happen to be active at the same time. Let  $Active(m, t)$  be a predicate to express that modality  $m$  is being used at some instant  $t$ . The simultaneous use of modalities of a set  $M$  over a finite temporal window  $tw$  can be formally defined as:

$$Parallel(M, tw) \Leftrightarrow (Card(M) > 1) \wedge (Duration(tw) \neq \infty) \wedge (\exists t \in tw \cdot \forall m \in M \cdot Active(m, t))$$

where  $Card(M)$  is the number of modalities in set  $M$ , and  $Duration(tw)$  is the duration of the time interval  $tw$ .

Sequential events may have to occur within a temporal window to be interpreted as temporally related. If they occur outside this window, then they may be interpreted differently. Modalities are used sequentially within a temporal window if there is at most one modality active at a time, and if all of the modalities in the set are used within the temporal window:

$$Sequential(M, tw) \Leftrightarrow (Card(M) > 1) \wedge (Duration(tw) \neq \infty) \wedge (\forall t \in tw \cdot (\forall m, m' \in M \cdot Active(m, t) \Rightarrow \neg Active(m', t))) \wedge (\forall m \in M \cdot \exists t \in tw \cdot Active(m, t))$$

### ***Equivalence***

We say two modalities are equivalent if separately those two modalities are necessary and sufficient for reaching a state. Equivalence express then the choice of the modality to reach a certain state.

$$Equivalence(s, M, s') \Leftrightarrow (Card(M) > 1) \wedge (\forall m \in M \cdot Reach(s, m, s'))$$

### ***Assignment***

We are talking about assignment when to reach a certain state, the user has not the choice of choosing the way he wants to interact. A modality is *assigned* to a state  $s$  to reach a state  $s'$  if no other modality can reach  $s'$  from  $s$ . In contrast to equivalence, assignment expresses the absence of choice: either there is no choice at all to get from one state to another, or there is a choice but the agent always opts for the same modality to get between these two states. Thus we can define two types of assignment:

$$StrictAssignment(s, m, s') \Leftrightarrow Reach(s, m, s') \wedge (\forall m' \in M. Reach(s, m', s') \Rightarrow m' = m)$$

$$AgentAssignment(s, m, M, s') \Leftrightarrow (Card(M) > 1) \wedge (\forall m' \in M. (Reach(s, m', s') \wedge (Pick(s, m', s')) \Rightarrow m' = m))$$

Equivalence and assignment both measure the choice available at some point in the interaction trajectory. Redundancy and complementarity go one step further by considering the combined use of multiple modalities under temporal constraints.

### *Redundancy*

We talk about redundancy if to reach a state, two or more modality must be used almost simultaneously to reach a state. We have to notice that if two inputs are using the same human resources, redundancy is impossible. And most of the time complicated due to the cognitive workload.

$$\text{Redundancy}(s, M, s', tw) \Leftrightarrow \text{Equivalence}(s, M, s') \wedge (\text{Sequential}(M, tw) \vee \text{Parallel}(M, tw))$$

This means that two modalities can be used at the same time in order to increase the « understanding » of the system.

### *Complementarity*

We say that an interaction is complementary if two or more modality must be use in a complementary way to reach a state. For example, if we'd like to do the action times 2. Operator times can be specified by hand gesture and operand 2 by pen. Function and arguments have to be specified by different modalities

$$\text{Complementarity}(s, M, s', tw) \Leftrightarrow (\text{Card}(M) > 1) \wedge (\text{Duration}(tw) \neq \infty) \wedge (\forall M' \in \mathbf{P} M (M' \neq M \Rightarrow \neg \text{REACH}(s, M', s'))) \wedge \text{REACH}(s, M, s') \wedge (\text{Sequential}(M, tw) \vee \text{Parallel}(M, tw))$$

## 1.1.4 CARE-Like properties of the user

A great advantage of multimodality as we talked before, is that the user can choose the modality he wants to use to communicate with the computer. If for example the user is occupied with his hand he would prefer use speech to communicate. We refer those preferences as U-preferences [NIGA 95a].

If only one modality is acceptable to the user, or if he has a strong preference for one particular modality, then we have a case of *U-assignment*. If there exists a subset of the possible modalities which he prefers to all others, but between which he is indifferent, then we have a case of *U-equivalence*. If the user prefers to employ two or more means of communication to convey the same information, then we have a case of *U-redundancy*. and if the user's preference is choose one modality for one aspect of the task and another modality for another aspect, then we have a case of *U-complementarity*.

The goal is to have a compatibility between the system and the user-preference (U-CARE properties). It should exist at least one modality which is acceptable for the user and the system. Then we pose the condition of a fitting between the system and the user expectations.

- For *U-assignment* : The system should have the same modality as the user wishes.
- For *U-equivalence* : The set of modality of the system should encompasses at least the modality with the one the user wants to interact.
- For *U-Redundancy* : the conditions are the same as U-equivalence.
- For *U-Complementarity* : Actions on which complementarity are possible as to work with same modalities as user wants to interact.

### 1.1.5 CARE properties and CARE-like properties : Conclusion

An important conclusion is that neither the properties of the system alone, nor those of the user alone, determine usability. System modelling can determine the properties of the system, but to understand those of the user, and hence usability, we need to turn to user modelling. It's then important to have both the informations about user preferences and system constraints. Firstly for not developing modalities that won't be used and secondly to avoid being in front of unavoidable constraints for the system.

The current definitions of the CARE properties provide a formal framework for reasoning about the design of multimodals systems.

### 1.1.6 Gesture

#### Definition :

The gesture term evoke the term of movement. But more specificity the movement of the above members : hand or head in order to execute a task or express a emotional state [BOUI 02].



**Figure 2: Pen-based gesture**



**Figure 3: Hand-gesture**

According to this definition, the main purpose of a gesture, is to achieve a certain task. This is exactly what we will try to accomplish in this thesis : allowing users to make gestures which will be interpreted by the computer that will accomplish the corresponding tasks. This will be accomplished with the help of two gesture recognizer : Quill and HandVu whom the explanations will be provided on chapter 2. Two kind of gestures will be interpreted in this thesis : pen-based gestures and hand-based gestures.

#### *Pen-based gesture :*

We define the pen-based gesture, the marks entered with a stylus or a mouse to invoke commands. We are using here, only single strokes gestures. It means that the stylus is never lifted up from the table to invoke a simple command. A single-stroke gesture is a single-path gesture that is one stroke. Thus drawing "L" is a single-stroke gesture, while "X" is not [RUBI 91]. So we use a graphical tablet to catch all the incoming gestures invoking commands. How we recognize and process them will be discussed in the following chapters.

### ***Hand-based gesture :***

An even more natural modality is to show to a camera a gesture especially done with the right hand. The different positions of the hand and the fingers will imply to invoke different commands. Here is an interesting comment from the biologist community about why hand gesture is one principal communication way.

*« People frequently use gestures to communicate. Gestures are used for everything from pointing at a person to get their attention to conveying information about space and temporal characteristics [KEND 90]. Evidence indicates that gesturing does not simply embellish spoken language, but is part of the language generation process [MCNE 82].» [NetLink01]*

Here we can find that principal language mode are obviously the spoken language but also the hands language, and that's why this two modalities appear as a good communication way with the computer. In this work we only developed gesture recognition without glove or any special devices, only the right hand. How we recognize those gestures and how we process them will be discussed in the following chapters.

## **1.1.7 Gesture recognition**

### **Definition :**

Gesture recognition involves determining the movement of a user's hand, arm, head or body through the use of a camera, or through a device with embedded sensors that may be worn, held or body-mounted [GART 06].

Due to gesture recognizers interacting more naturally with the interfaces is now possible. A short state of the art can show the Nintendo Wii.



***Figure 4: Wii plateforme***



***Figure 5: Wii wheel  
contrôleur***

The gestures recognizers we used to accomplish multimodality are Quill and HandVu. Specifications of those recognizers are provided on chapter 2.5 and 2.6

## 1.2 Motivations

Our motivations for developing multimodal interfaces are double. Firstly, there's a potential commercial issue and secondly, multimodal interfaces should contribute to increase the accessibility, the usability and the robustness of any computer interfaces.

### 1.2.1 Potential commercial issue

According to Gartner Inc. Figure 6 and Figure 7 [GART 05][GART 06], gesture recognition could have a bright future. Gartner Inc. developed the «Hype cycle» which characterise any technological invention in terms of potential industrialisation. Before reaching an industrialisation state, a new technology has to step different phases which bring them closer to the market. The first step of this hype cycle is called «Technology Trigger» which is : the technology trigger, or breakthrough, product launch or other event that generates significant press and interest. Gesture recognition are now close to the end of this phase which is quiet motivating because in that same report of last year, gesture recognition were at the beginning of that phase. The following phase of this cycle is the «peak of inflated expectations» which is a phase during which a lot of expectations and frenzy of publicity tend to imagine unrealistic expectations but successful applications can be developed even if most of the time failure occurs. This means that work which has been done for the gesture recognizers have not been abandoned. Tough market adoption of this technologies is in 5 to 10 years and market intrusion are about 1 % of the target audience, this technology is still considerable. So we can think that this field of study will keep on interesting industries and research.

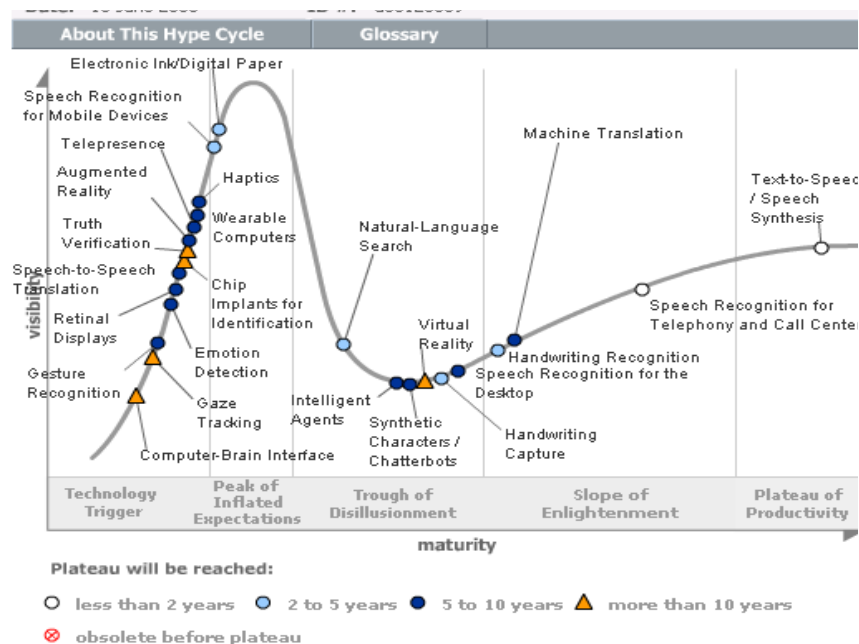


Figure 6: Gartner Hype Cycle 2006 [Gar 06]

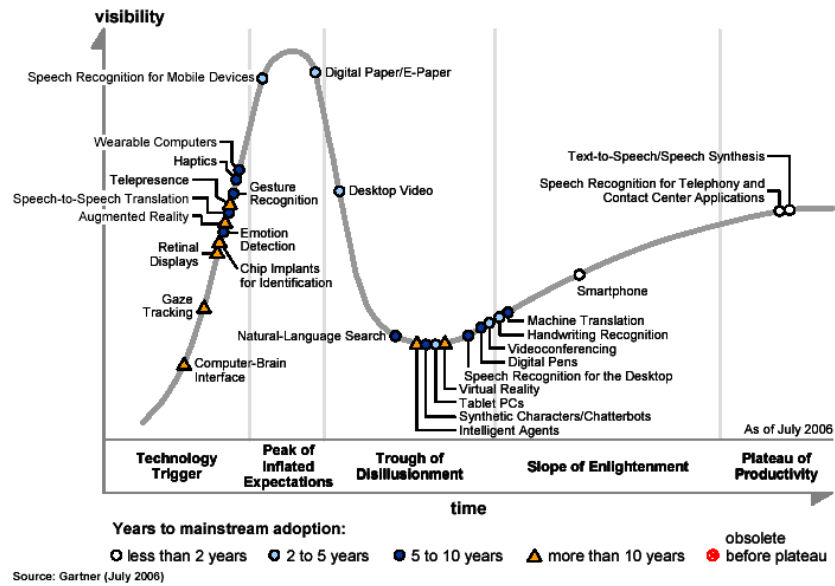


Figure 7: Gartner Hype Cycle 2006 [Gar 06]

## 1.2.2 Considerable ameliorations of interfaces

Most people who interact with computers spend most of their time entering information [CARD 80]. Due to this input bottleneck, the total time to do many tasks would hardly improve even if computers became infinitely fast. Thus, improvements in input technology are a major factor in improving the productivity of computer users in general [RUBI 91].

The communication between man and machine will become more and more present in the future, the need to reduce the complexity and to increase the naturalness of this communication is real. Introducing and combining new input technologies would increase the computer interfaces usability, accessibility and efficiency.

Allowing computer to understand more concise and more powerful information due to a better understanding provided by any user would increase the efficiency while providing choices of modality would improve accessibility and usability.

## 1.3 Goals

The final goal of this thesis is to integrate the multimodality on two platforms. Our modalities will be the pen-based gesture and hand-based camera recognition. We will then realize a comparative study between these two platforms.

More precisely, Our goals are to integrate this multimodality on two platforms, InterpiXML and OpenInterface that we will introduce in the following chapters. We will integrate both modalities on both platforms.

To achieve this we will modify the InterpiXML architecture to be aware of pen-based gesture and of natural hand gestures.

For OpenInterface, we will develop two generic components, one for the pen-based gesture recognition and one for the hand gesture recognition. This genericity will enable OpenInterface to reuse those components for any application.

When those modalities will be integrated, we will evaluate the two platforms and compare them. This comparison will be assessed in terms of CARE properties but also in terms of utilisability based on the IBM forms. For reaching this final goals we will introduce an experiment we performed.

## 1.4 Reading plan

After this introduction, where we introduced the multimodal interfaces, their advantages and defined the main concepts of the present thesis we will introduce the skeleton of this thesis.

Chapters 2 will focus on the state of the art of this field of study of the human-computer interaction. First, we introduce the current existing gestures for pen and hand-based recognition (2.1). Then we discuss about the gestures qualities (2.2). Section 2.3 present actions set on interfaces. In section 2.4, we present some existing toolkits for pen and hand gestures and introduce in section 2.5 and 2.6 toolkits that we have chosen and explain why we choose them. After talking about the specification language choice : UsiXML (2.7), we will finally introduce one existing multimodal platform named OpenInterface (2.8).

The chapter 3 will last on the conception itself. We present there all our design choices.

The chapter 4 focus on the integration of multimodality to InterpiXML platform. Integration of the pen-based recognizer and hand gestures recognizer. We will explain the architecture and implementation for each modality and show an example and provide an evaluation of InterpiXML upgraded with multimodality.

Chapter 5 focus on the development of components for OpenInterface. Those components are a gesture pen-based recognizer and a gesture camera recognizer. We will also explain how it works and show example and provide an evaluation as we did for InterpiXML.

Chapter 6 will provide the evaluation of the experiment we made in order to compare OpenInterface and InterpiXML platforms and also a comparison of the integrated modalities. The results should provide good advices for future work.

We will finish with conclusion where we show all work that we done, and give idea for future work.

Thanks for your active reading.

## 2. State of the art

Here we will introduce to different softwares, environments and techniques used or re-used during conception and implementation. We first present the current existing pen and hand-based gestures. Then we will define the gesture qualities. Section 2.3 will focus on actions set on interfaces. Then we will give some existing toolkits and chapters 2.5 and 2.6 will describe the specific toolkits we choose. After we will discuss about the specification language choice and finally describe the OpenInterface platform on chapter 2.8.

### 2.1 The current existing gestures

Here we will present some applications which use pen based gestures or hand camera based gestures. This is a short state of the art which show how gestures recognition is present and could be more present in our every day life. But user have to keep in mind the different field where pen-based is present such as such as text editing, sketch, modelling, UI design, 3D manipulation and navigation, etc.

#### 2.1.1 Pen Based Gesture :

##### **All in one gesture plug in for Firefox :**

The all in one gesture plug-in for Firefox enable to use different mouse gestures in order to invoke commands. Those commands are numerous it goes from going to the previous page to open all links in pages and navigate between tabs. Those kind of mouse gestures showed an improvement of the speed navigation and a great satisfaction of the users [MOYL 02]. you can find all the possible actions here : [NetLink02] and a video here showing the plug-in in action [NetLink03].



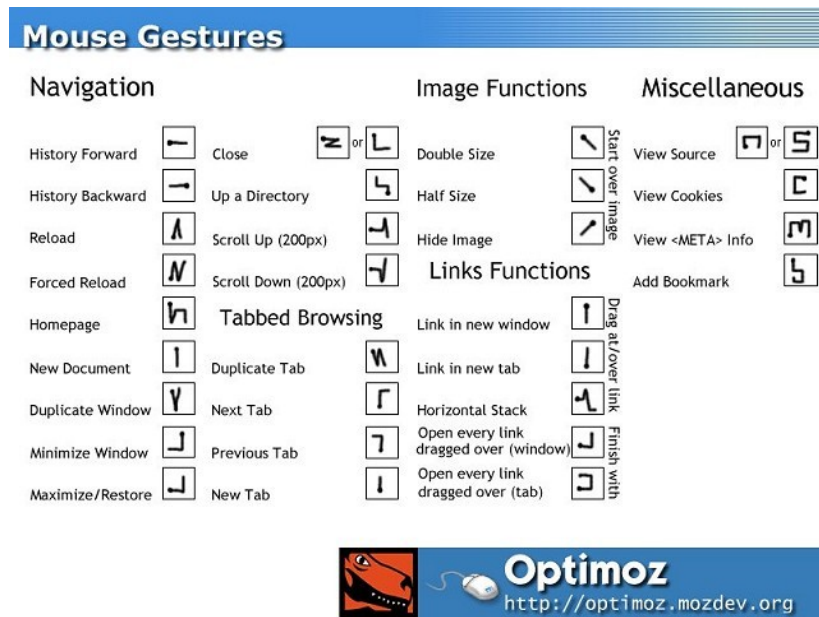


Figure 8: Set of possible actions with gesture mouse with All in one gesture plug in for Firefox

### Matis system :

MATIS is a multimodal system of information on air transports. It provides, in response to requests of the user, informations on the flights between two cities. The system was developed on NeXT machine with the system of voice recognition Sphinx. MATIS authorizes statements of orders such as the sentence known as "I would like a U.S. air flight from this city to this city" combined with two selection-mice to specify the towns of departure and arrival. MATIS allowed a study on the software architecture of the multimodal systems like on the integration of the methods: mechanism of fusion of the multimode events and references. [NIGA 95b].

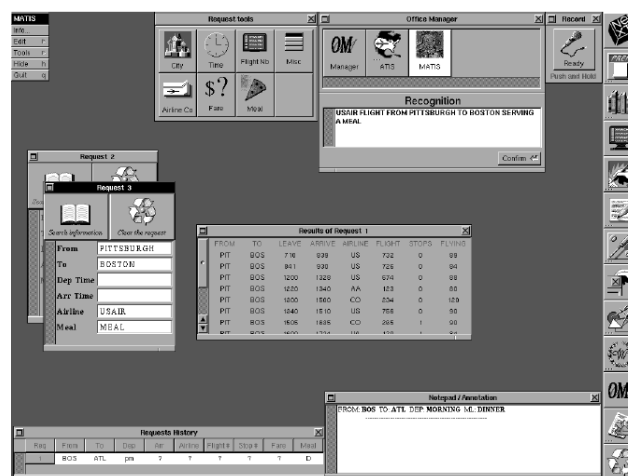


Figure 9: The MATIS application

### Handwriting recognition.

Handwriting recognition system such as graffiti is also an application of pen-based gestures. The graffiti alphabet has been developed for the PALM platform for recognizing gesture as characters or numbers. Graffiti alphabet is composed only with one-stroke gestures. [NetLink04]. In fact Graffiti's gesture are used with the specific recognizer which recognize up to 97 % of the gesture after a few training test of the user [MACK 97].



Figure 10: Graffiti alphabet

### Operating Systems :

The accomplishment of all the previous applications are now integrated to Operating systems. For instance both of the OS leader Microsoft and Apple provided their OS developed for pen recognition. With the Microsoft Windows XP tablet PC edition [NetLink05] and the Mac OS X Tiger[NetLink06]. This is the accomplishment of pen based application. But our goal is not the integration of a single component such as the pen-based application. Our goal is to provide multi modality as we said before.

### 2.1.2 Hand Gestures

Hand gestures can take a lot a different positions. To try to characterize each positions, some language description have been realized. A well-known is described on [MONE 06]. Language describe the hand position and the features of each finger. Another system is to realize gesture sample where each hand gestures is named. The following gestures are currently defined (right hand shown) :



Fist



Index finger point



Up Yours (Middle  
finger point)



Two fingers point

			
Ring finger point	Ring-index finger point	Ring-middle finger point	Three finger point (or not little finger point)
			
Little finger point	Howzit (index and little finger point)	Little-middle finger point	Not ring finger point
			
Little-ring finger point	Not up yours	Not index finger point	Flash hand

**Figure 11: Hand Gestures illustration**

We see with this tab that some gestures are really hard to do. For example, the Ring finger point need practise to realize it correctly.

At the moment no much professional software really use hand gesture recognition for industrial applications. Specially in our case of 2D real-time recognition software for only one hand. But some researcher works on this to try to improve hand tracking and gesture recognition.

One of the best has been developed at the School of Computing in Dublin City University, Ireland. This is a hand gesture recognition system for replacing the mouse. So you can move the cursor across the screen and realize right and left click only with the index finger. Other toolkits recognizer will be presented on section 2.3.



*Figure 12: Hand gesture recognition system for replacing a mouse*

## 2.2 Gestures Qualities

Gestures are not only unspecified marks or insignificant hand gestures. Good gestures need to have some properties. Although both pen based and camera based gestures need to meet some some qualities we will separate the explanation of those qualities in different points. A lot of experiments ([LONG 99][TIAN 06][LONG 01][LONG 99b]) have been made to find how to design better and to find what are the factors increasing the quality of those gestures.

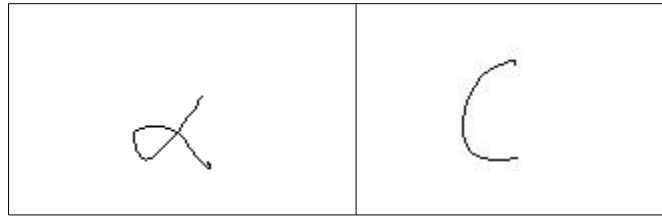
### 2.2.1 Pen based gestures qualities

#### **Iconicity**

When humans are communicating, they are using gestures to increase the understanding of the listener and obviously, the gesture usually means what the speaker is saying. Iconicity principle is based on this. It means that gestures that are designed are close to the interpretation of this reality.

Iconicity : «memorable because the shape of the gesture correspond with is operation» [LONG 01b].

For example, Figure 13 represent the action of delete because it represent a pair of scissors and the action of copy because de «C» stands for Copy.



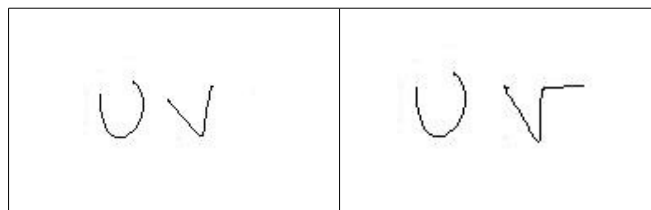
*Figure 13: Delete and Copy iconic gestures*

### **Learnability**

Another important gesture quality is its learnability. Users sometimes forget gestures because they are numerous or because they are complex or even because they are not iconic. 90% and more participants held that pen gestures with visual meaningful related to commands are easy to be remembered and learned.[TIAN 06]. An alternative taught by Tian [TIAN 06] to increase the rememberability of the users was to represent gesture as the first character of the command name (i.e a c for copy ). It could be an alternative but you can't then have any characters in your application except if for example only characters are possible in some areas of the application. If users spend their time for checking which gesture is convenient for executing a command in the manual, user will get bored soon. So gestures have to be easily remembered.

### **Gesture recognizer recognizability**

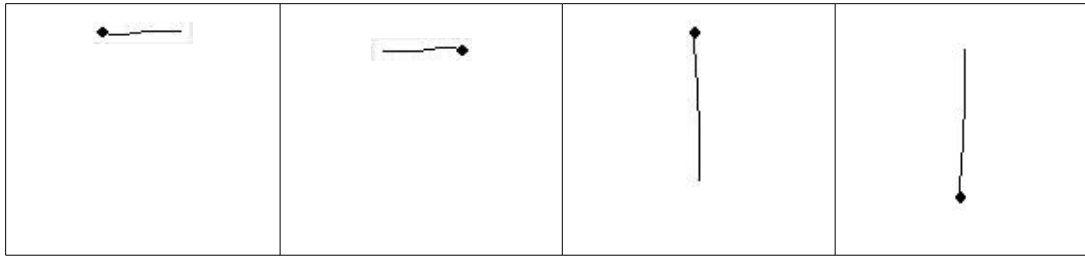
This paragraph is an non-sense if the recognizer as been created for recognize some specific gesture. But in this thesis we will use a toolkit named Quill which recognize gestures which are created for any applications. Naive gestures designers often created gestures that the computer viewed as similar and thus were difficult to recognize [LONG 01b]. Sometimes, there is a trade-off between improving the gesture recognizability for the gesture recognizer and decreasing the recognizability for the user. See on figure 14 left, this kind of gestures can easily be confused by the recognizer. But on figure 14 right, the recognizer will increase the recognizability. So we have to found a balance both for the user and recognizer.



*Figure 14: u and v different design*

### **Compatibility and coherence**

Gestures also are better learned and better used when they are compatible and coherent. Gestures are best perceived when they are introduced in a uniformed and stable way. See in figure 15, it's implicit for the user if the left direction is go to the left that the right direction will be go to the right. This is an illustration of coherence and compatibility.



*Figure 15: Example of gesture coherence.*

If the first gesture means «going to the right» it's implicit for the user that the second gesture will mean «going to the left» just as if the third gesture means «going down», the fourth means «going up».

### 2.2.2 Hand based gestures qualities :

We will here evaluate the hand gestures qualities on the four same criterion as upside and try to characterize what's a good hand gesture.

#### Iconicity

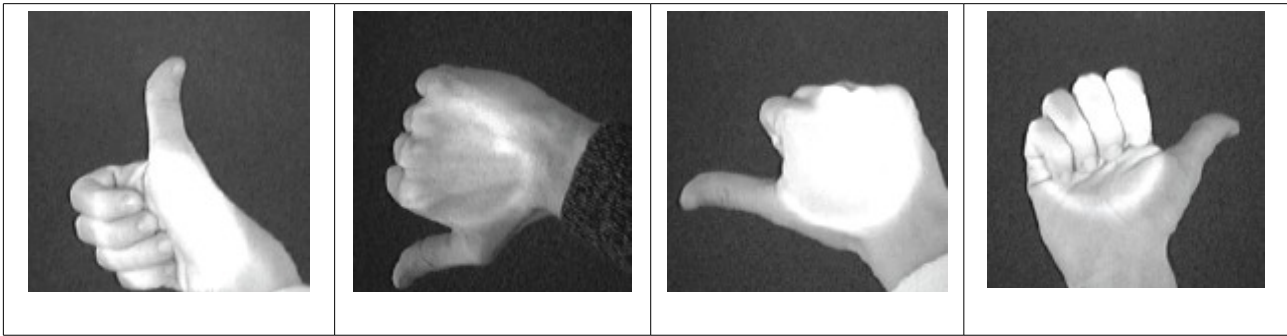
For hand gestures, it's used to find gesture which reflects action associated. For example, a closed hand to close a frame. Or as in this example a thumb up signify a validation action. Everybody agree that this gesture means an agreement.



*Figure 16: Thumb up for Ok*

#### Learnability

A good iconicity gesture is also good for the learnability. It's easier to remember gesture that we know before use the modality and which have same sense that in life. To expand the capacity that people have to learn gestures, it's a good things to have opposite gestures for opposite actions or similar gestures for similar actions. As for indicate direction, if the person know that the direction have to be indicated with the thumb (One information to learn), the person can easily do four actions (Up, Down, Left, Right).



*Figure 17: One information to learn to do four actions (Up, Down, Left, Right)*

### **Gesture recognizer recognizability**

To easily recognize hand gesture by specific software it's better to have gestures very different. But with the hand it's not very easy to change the hand morphology. You can rotate the hand, close some fingers, ... but it appear quickly that the number of different postures are limited or begins to be too similar and the recognizer begins to have poor recognition rate. So Limited number of different gestures is a way to improve the recognizer recognition rate.

### **Compatibility and coherence**

As explain a few in the iconicity and learnability sections, similar gestures have to be chosen to do similar actions and opposite gestures for opposite actions. The previous example with direction gestures is also a good example of compatibility and coherence. The hand doesn't change because it's the same action (indicate a direction), the hand just turn to indicate the specific direction with the thumb.

## **2.3 Actions set on interfaces**

We divided into four main parts the different actions we found. In this dissertation, we will only interest in those actions : The Windows managing actions, the browsability or navigation actions, the validations actions and the characters and numbers insertion. We will introduce here the commands we will use in this dissertation.

### **Windows managing actions**

Represent typical actions for close, reduce or minimize/maximize a windows. These actions have their representations in the high-right corner of almost all windows using Microsoft Windows.



*Figure 14: Graphical window managing actions representation*

- Close : Close the window (*Alt+F4*)
- Reduction : Reduce window in the task bar. (*Alt+SpaceBar* then *u*)
- Minimization/Maximization : Minimize the window if it's in full screen. Maximize it in full screen otherwise (*Alt+SpaceBar* then *r* and *Alt+SpaceBar* then *n*).

### **Browsability actions**

- Next item : Put focus on next item in the current window (*Tab*)
- Previous item : Put focus on previous item in the current window (*Shift+Tab*)
- Up : Select item upside, if possible. Different behaviours depends on item type (*upArrow*)
- Down : Select item downside, if possible. Different behaviours depends on item type (*DownArrow*)
- Right : Select item rightside, if possible. Different behaviours depends on item type (*RightArrow*)
- Left : Select item leftside, if possible. Different behaviours depends on item type (*LeftArrow*)

### **Validations actions**

- Selection : Select the current item or click on button (*SpaceBar*)
- Reset : Reset the current item if that's a text field

### **Characters and numbers**

Filling fields is also possible with insertion actions.

- Characters : We also integrated for pen-gesture the characters from a to z.
- Numbers : We also integrated for pen-gesture the characters from 0 to 9.

## **2.4 Existing Toolkits**

We will here present some existing toolkits for pen and then hand gesture recognitions. We will then describe and argue in the two next chapters (2.5 and 2.6) why we decided to use Quill and HandVu toolkits for our developments.

### **2.4.1 Pen-based Toolkits**

#### *PenBuilder*

PenBuilder is a toolkit for developing pen-based user interfaces. This toolkit employs Pen-UI orientated event model, rendering model and interaction semantic model. The attributes of ubiquitous computing and using ink as a first-class data type were addressed in the design of this toolkit.



PenBuilder provides both hierarchical and flat structures for manage graphical objects. Both heavyweight and lightweight components are enabled for simplifying developing interface and for improving performance. An event parse tree is devised for parsing low-level pen input and generating high-level interaction events for applications. Some facilities for ink manipulation and rendering were avoided. The first version of this toolkit was built in 1999

Disadvantage of this is there we didn't found any documentation or sources of this toolkit.

### *Microsoft XP tablet Edition development kit*

The Windows XP Tablet PC Edition Software Development Kit facilitates building ink-enabled applications for Tablet PC. The combination of software and hardware in a Tablet PC enables these methods of user interaction and allows for a rich, interactive, and productive computing experience for users.

The Tablet PC platform encompasses Windows XP and its extensions that enable input and output of handwriting and speech data on a Tablet PC as well as interchange of this data with other computers. The Windows XP Tablet PC Edition Software Development Kit (SDK) enables you to build ink-enabled, pen-enabled, and speech-enabled applications and supporting hardware for the Tablet PC.

However, this only works with Microsoft XP tablet edition and more we can't define our gestures. It could have been useful for character recognition.[NetLink07]

## **2.4.2 Hand-based Toolkits**

Minority Report movie inspired few developer in the hand gesture recognizer research. The movie show Tom Cruise who move pictures displayed on a screen only by hands moving. We describe here some existing system for hand-computer communication through camera or webcam. Our final choice is the HandVu software developed on the University of Santa Barbara . And we will describe why we choose this toolkit on section 2.5.

### *Isight Sony*

The latest Apple laptops and iMacs have another hardware feature that could be used as an human interface device: their built in iSight camera. Rather than simply monitoring light levels, the iSight creates high resolution video information that software can analyze for movement, such as hand gestures. The idea isn't new. In 2003, Sony released a camera unit for the PlayStation 2 called EyeToy that detects colour and movement to involve players in a game. Players stand in the active area in front of the camera, and jump, kick, and punch to trigger actions in the game. Games range from Groove, a dancing game that helps burn the calories off fat kids, to Operation Spy and other interactive games that simulate moves from karate, bowling or volleyball.

A common problem related to the Sony EyeSight involves inadequate lighting in the play area. MacBook and iMac users are illuminated by their display, and will generally be sitting closer and centered in front of the camera, making it easier to develop a standard set of gestures that are easy to recognize. Another problem is that this toolkit is not open Source and furthermore not free.

### ***Georgia Tech Gesture Toolkit (GT2K)***

The Georgia Tech Gesture Toolkit (GT2k) provides a publicly available toolkit for developing gesture-based recognition systems. The Georgia Tech Gesture Toolkit GT2k leverages Cambridge University's speech recognition toolkit, HTK, to provide tools that support gesture recognition research. GT2k provides capabilities for training models and allows for both real-time and off-line recognition [WEST 03].

## **2.5 Quill Toolkit**

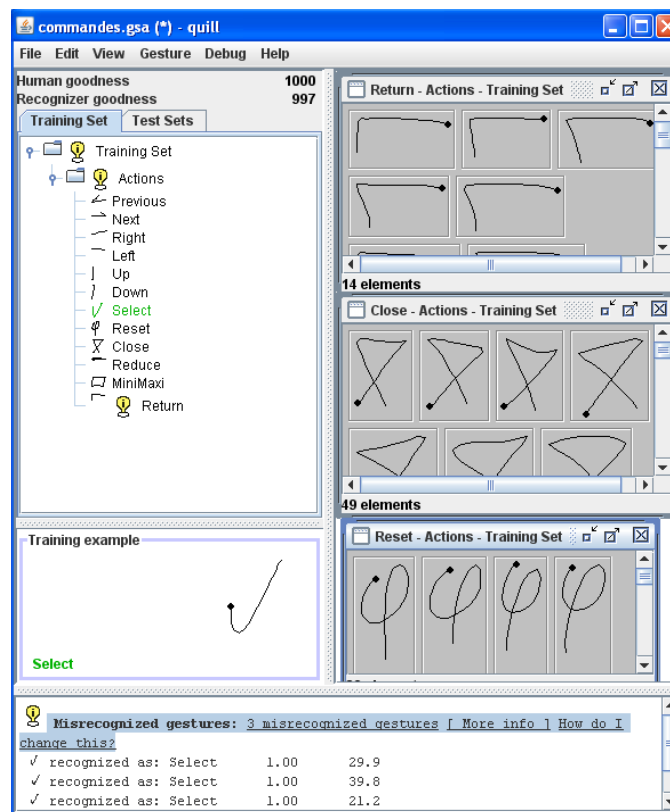
### **Introduction**

Quill is a toolkit created to help designers of pen-based user interfaces to create better gestures. Quill is based on Rubine algorithm that we will briefly introduce in this chapter. It has been developed by Allan Chris Long for his Phd Thesis in computer science for Berkley in 2001 [LONG 01b]. [NetLink08]

### **General principle**

To use the Quill toolkit, first you have to draw into different gesture category all the gestures you want to get recognized in the future. In fact, a gesture category is a set of same gestures. The idea of the gesture category is to gather informations about a same gesture which can't be drawn exactly twice the same. So that, you have to draw the same gesture a few into each gesture categories (10 to 15 times). During this phase called «learning phase», the great advantage of Quill take place. In fact, Quill informs the user instantaneously of the possible similarity of the new drawn gesture category and the existing gesture categories. This similarity is computed either for the recognizer, we mean for avoiding the bad recognition, than for the supposed human perception and give also advices on how to recover from those eventual problems. They mean by human perception, the facility to remain the gesture and not confuse them with another gesture.

When the gesture has to get recognized, the recognizer is called and compute once again the specific features of the drawn gesture and compare them with the values of the features of the «learned gestures». The recognizer returns the list of all the gesture following a decreasing order of an indice of similarity. The first item in this order is then the more-look-like gesture based on the proximity of the features values.



**Figure 18: Quill illustration :**

We see on the left the set of gestures, above the metric of the recognizer potential recognition (here 997) and the human perception of similarity (1000). Also on the right, the gesture design.

### Rubine's algorithm :

As we said, Quill toolkit is based on the Rubine algorithm which has been developed by [LONG 01b] for his Phd Thesis. Rubine's is what we call a feature-based algorithm. In the field of pen based gestures recognition, it also exists algorithms based on neuronal networks those two algorithms are the most commons. According to [LONG 99], neuronal-networks algorithms have a high recognition rate but need a long training time while feature-based algorithms have a lower recognition rate but have more advantages : the number of training examples is small (10 to 15), easy to implement and others system using it has been successful [CHAT 96][FRAN 95]. The fact that it only require few drawing example is considerable because as we will design our gestures, we don't want to spend our time in drawing.

The principle of a feature-based algorithm is that some features (11 in Rubine's algorithm and 16 in Quill) will be computed (for exemple the distance between the beginning and the end of the gesture) for each gesture. During this learning phase, when a new gesture is drawn, the algorithm will compute the average value of that gesture in term of feature (taking into account the uniform distribution of this feature). So each group of gesture (for example all the gestures representing a "c") will have a value for each feature. When a gesture has to get recognized, the algorithm compute the same features for the new drawn gesture and recognize it has the gesture category having the nearest values for those features.

### **The specifics features :**

The features for Quill are computed are listed below :

- Bounding box. The bounding box for a gesture is the smallest upright rectangle that encloses the gesture.
- Cosine of the initial angle
- Sine of the initial angle
- Bounds size. This feature is the length of the bounding box diagonal.
- Bounds angle. This feature is the angle that the bounding box diagonal makes with the bottom of the bounding box.
- Ends distance. This feature is the distance between the first and last points of the
- Ends angle cosine.
- Ends angle sine.
- Total length of the gesture.
- Total angle. This feature is the total amount of counter-clockwise turning. It is negative for clockwise turning.
- Total absolute angle. This feature is the total amount of turning that the gesture does in either direction.
- Sharpness. This feature is intuitively how sharp, or pointy, the gesture is. A gesture with many sharp corners will have a high sharpness. A gesture with smooth, gentle curves will have a low sharpness. A gesture with no turns or corners will have the lowest sharpness.

### **Advantages of Quill**

Maybe we could have used other toolkits such as gdt, which is the ancestor of Quill, or Agate [LAND 93] but we will try to explain here why we choose Quill regarding to his advantages.

Quill enable us to define our gesture. It means that it's not only a gesture recognizer but also a tool for design gesture that will be later recognized by Quill. This is an important point because the application on which we will add pen-based recognition use action that are not all defined in the literature (for example resetting the content of a text field).

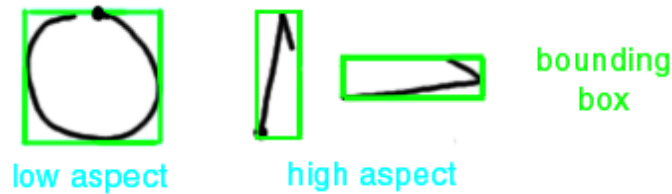
Quill is a tool that prevent user from designing wrong gesture at 2 sights. In fact, when the user is designing gesture during the learning phase, Quill automatically compute a recognition rate (recognizer goodness) which inform the designer of potential misrecognition if that value is too low (1000 is maximum).

Moreover information about potential misrecognition Quill also provide an estimation about the quality of the gesture in term of human learnability and similarity for humans (human goodness) that correlate with 0.56 [LONG 01] of the results.

Features for estimating this criteria are :

### Aspect

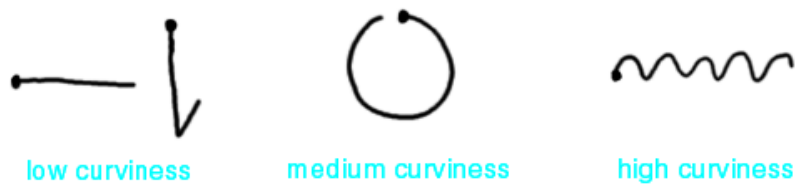
This feature is the extent to which the bounding box differs from a square. A an example with a square has bounding box aspect of zero.



*Figure 19: Aspect feature*

### Curviness

This feature is how curvy, as opposed to straight, the gesture is. Gesture with many curved lines have high curviness while ones composed of straight lines have low curviness.

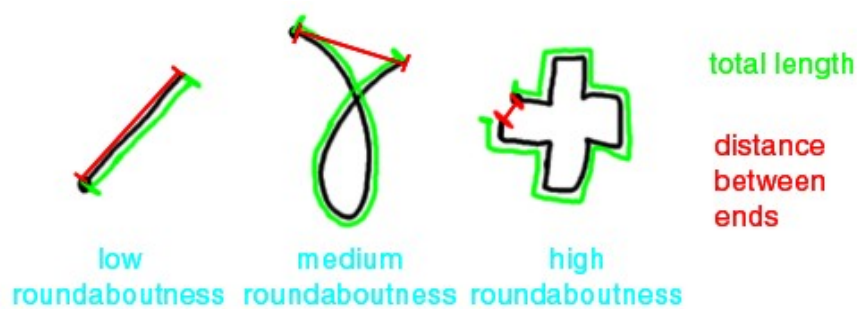


*Figure 20: Curviness feature*

A gesture with no curves has zero curviness. There is no upper limit on curviness.

### Roundaboutness

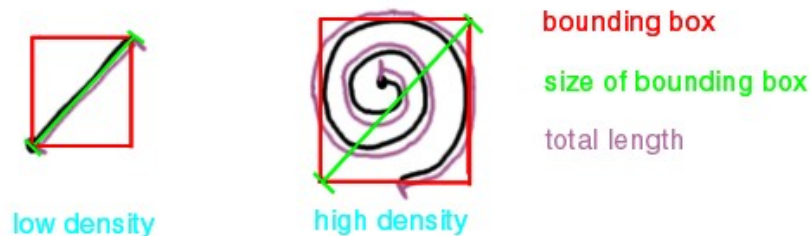
This feature is the length of the gesture divided by its endpoint distance.



*Figure 21: Rondaboutness feature*

### Density

This feature is how intuitively dense the lines in the gesture are. Formally, it is the length divided by the size of the bounding box.



*Figure 22: Density feature*

The lowest value it can have is 1. There is no upper limit.

Those features are responsible for the potential similarity for humans. So when a new gesture category is drawn we have an immediate estimation about the human recognizability. The metric used is the human goodness metric which goes to 1000. It's then up to the gesture designer to define a level of human goodness for his gestures.

Another advantage of using Quill is also that for avoiding misrecognition, we can accept the gesture recognition only if this indice of recognition is above a certain value. We can define a level of similarity which prevent us from misreconizing a gesture which is we think a key point in some critical jobs. We could even think of defining a level of similarity for some interpreted gestures and not for all the gestures. For example, closing a window is a more critical action than reducing the window (all the informations contained in the window would be lost). By this way, the user has just to do a kind of confirmation for this critical action.

Moreover, some other advantages where that it was really easy to integrate (due to a .jar file), the short tutorial we found on *sourceforge* is well done and easy to understand. But we have to remark that to be integrated to for example a Java application, we need another toolkit called Satin on wich we will talk a few in the next section.

### Satin

Our use of Satin was very poor so we will be short about it. It's a toolkit developed by the university of Berkley in 2002. Satin has been created for making effective pen-based application easier. The two facets of SATIN we used are :

- The integration of pen input with interpreters.
- The libraries for manipulating ink strokes.

So we've created a SATIN sheet on which the user can draw the gestures which are interpreted with the libraries that can handle the ink strokes. Then the interpreter is the one we described into the Quill section. SATIN then get back the recognized gesture of Quill. The advantage of such manipulation is that we can define has much interpreters as we want.

## 2.6 HandVu Toolkit

Why did we choose the HandVu hand-gesture recognition software ?

Obviously more than one hand gesture recognition software exist. If we took it, it's because it has been recommended by many subject specialists. HandVu is a research work developed in the University of California (Santa Barbara) and is completely free and open-source. Its principal advantages are :

- It works in real-time and without need for camera or user calibration.
- It's nearly not sensible to hand size and background.
- It works under Windows and Linux like InterpiXML and OpenInterface.
- It doesn't require other specific materials than a computer and a webcam (for example no color gloves).
- It's still improved by group of people. Next improvement announced will be : Tracking with higher frame rates on Linux – More recognized postures (picking posture) - High-precision pointing, tracking (same spot on hand across recognitions).

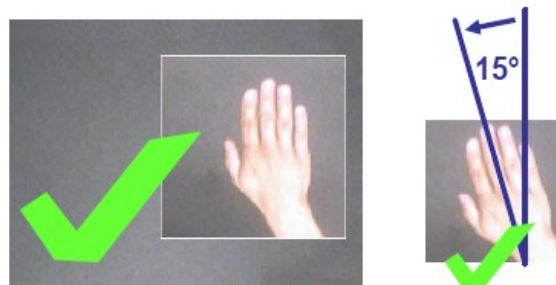
First HandVu detects the hand in a standard posture (close position) and then track it and recognize key postures. We show here principal key features (extract from software website [NetLink09] ).

### Camera :

HandVu work with a camera that views the space in front of a sitting or standing person from a top-down view. It should deliver at least a 320x240 resolution.

### Hand detection :

The hand is detected only in a standard posture and orientation entirely with respect to the camera, called the *closed* posture: recognized postures. This is necessary to avoid inadvertent gesture commands and to speed the image processing.




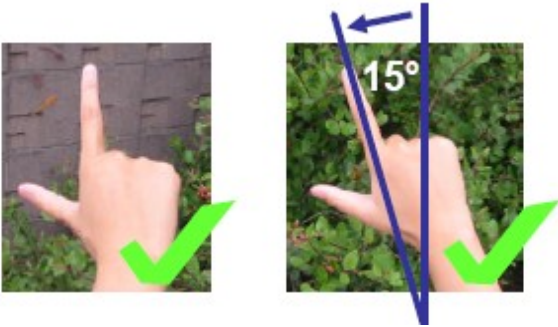
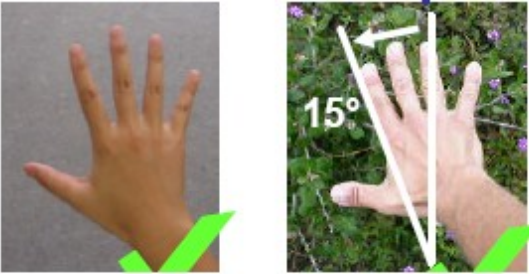
*Figure 23: Posture for hand detection*

### Hand tracking :

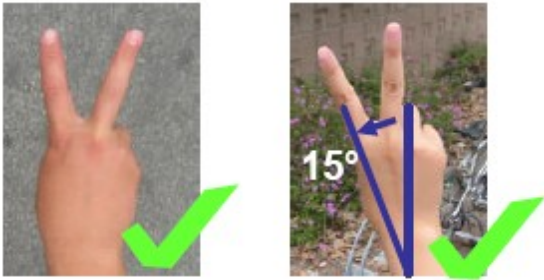
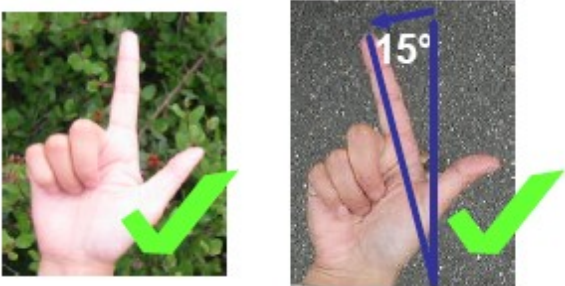
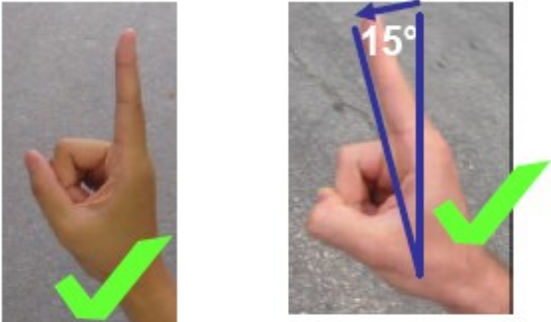
Once the hand has been detected, you can move the hand around in any posture. The better the lighting conditions are (uniform without harsh shadows) and the less brightness variation exists in the background, the better tracking will work. Avoid all too rapid movements or quick posture changes if you experience problems.

### Posture recognition :

All of the six recognized postures can be performed at any time during tracking and they will be recognized. Note that all postures are to be performed in a plane parallel to the imaging plane, facing upwards in the image, and with no more than 15 degrees counter-clockwise rotation (to the left). You will have to practice the gestures a few times until you achieve a good recognition rate.

Name	Posture
closed	
Lback	
open	



victory	
Lpalm	
sidepoint	

*Figure 24: Recognized postures*

#### Functionality :

- A "GestureServer" is automatically started and accepts TCP/IP connections on port 7045 and write GestureEvents in ASCII format when gesture is recognized.
- The key shortcuts 0,1,2,3 select different verbosity levels. If you are experiencing trouble with the recognition, please select level 3. The white rectangle is the initial detection area.
- Pressing "r" restarts the detection in the initial area.

## 2.7 Specification language choice

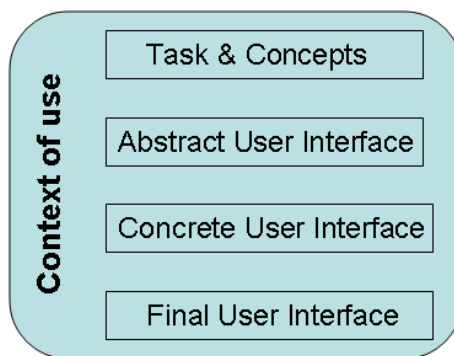
To develop multimodal interfaces we first need to choose a specification language for these interfaces. A lot of languages has been conceived with conciousness to easily develop new interfaces. Here we can talk about ximl, uiml or XISL. To conceive multimodal interfaces we need to decide which language to use to specify these interfaces. As first we decided to provide multimodality on the InterpiXML interpreter, we thus choose as work hypothesis to use UsiXML language to describe our future multimodal interfaces. We insist on fact that we here choose UsiXML as a work hypothesis. We will next present the language and its associated interpreter.

### 2.7.1 UsiXML

#### The language

Here we consider UsiXML (USer Interface eXtensible Markup Language), a User Interface Description Language that allows the specification of various types of user interfaces. According to [STAN 07] UsiXML has been selected due to the following motivations:

- UsiXML is structured according to the four basic levels of abstraction (Figure 25) defined by the Cameleon reference framework identified in [CALV 03]. This framework is a reference for classifying UIs supporting multiple target platforms or multiple contexts of use in the field of context-aware computing and structures the development life cycle into four levels of abstraction: task and concepts, abstract UI, concrete UI and final UI. The identification of the four levels and their hierarchical organization is built on their independence with respect to the context in which the final software system is used. Thus, the Task and Concepts levels is computation independent, the Abstract UI level is modality independent and the Concrete UI level is toolkit independent.

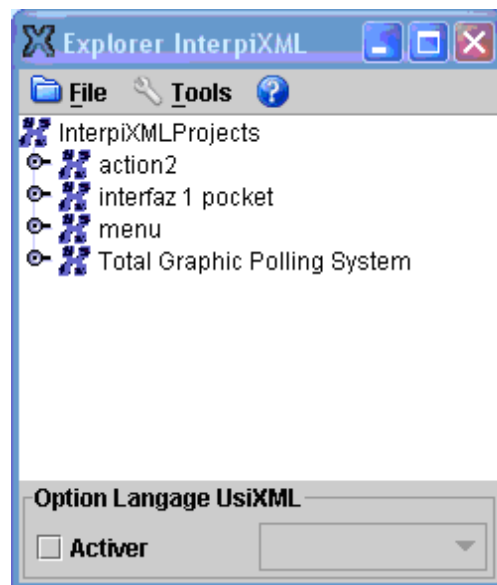


*Figure 25: The Cameleon reference framework for multi-target UIs*

- UsiXML relies on a transformational approach that progressively moves from the Task and Concept level to the Final User Interface
- The steps of the transformational approach define in a comprehensive way their logic and application [LIMB 04] (Requirement 9. Method explicitness).
- The transformational methodology of UsiXML allows the introduction of new development sub-steps, thus ensuring the possibility to explore alternatives for each sub-step and to add new ones (Requirement 10. Method extendibility)
- UsiXML has an underlying unique formalism represented under the form of a graph-based syntax. (Requirement 6. Ontology homogeneity)
- UsiXML allows reusing elements previously described in anterior UIs to compose a UI in new applications. This facility is provided by the underlying XML syntax of UsiXML which allows the exchange of any specification. Moreover, the ability of transforming these specifications with a set of transformation rules increases the possibilities for reusing them
- The progressive development of UsiXML levels is based on a transformational approach represented under the form of a graph-based graphical syntax. This syntax proved to be efficient for specifying transformation rules and an appropriate formalism for human use (Requirement 7. Human readability)
- UsiXML supports modality independence as UIs can be described at the Abstract UI level in a way that remains independent of any interaction modality such as graphical interaction, vocal interaction or 3D interaction (Requirement 4. Ability of modeling a UI independent of any modality)
- UsiXML supports the incorporation of new interaction modalities thanks to the modularity of the framework where each model is defined independently of the others and to the structured character of the models ensured by the underlying graph formalism (Requirement 5. Extendibility to new modalities)
- UsiXML is supported by a collection of tools that allow processing its format (Requirement 11. Machine processability of involved models)
- UsiXML allows cross-toolkit development of interactive application thanks to its common UI description format (Requirement 12. Support for toolkit interoperability).

### The interpreter : InterpiXML

InterpiXML is a runtime UsiXML interpreter for a Computer context of use. It works under Windows, Linux and Mac OS platform. It generate a Java Swing interpretation of user interface described in the UsiXML file. It encompasses an explorer where user can find his folder and UsiXML files.



*Figure 26: Explorer InterpiXML v1.0*

When we began development on it, it supported version 1.6.4 of UsiXML language. The different functionalities that it offers are :

- Interpret a UsiXML file with a double-click on in the explorer or with the open functionality on the *File* menu
- Register UsiXML file in the explorer
- List the preferred languages for interface rendering
- Change the presentation look&feel dynamically
- Choose the interface language (« *Option Language UsiXML* ») if language is described in the UsiXML file

## 2.8 OpenInterface

### Introduction :

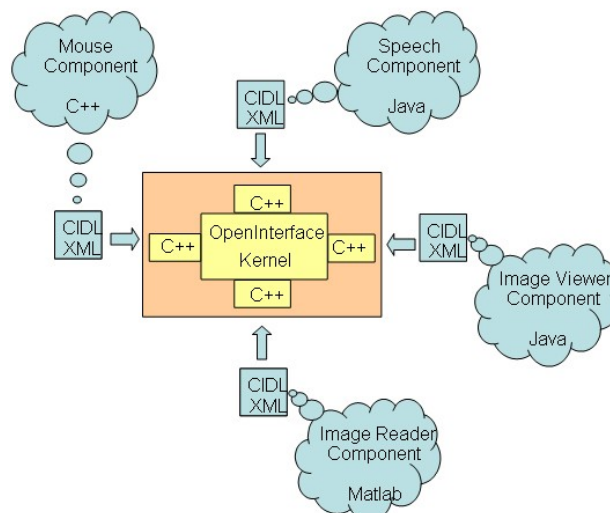
OpenInterface is a project that take his origin in the Similar Network of Excellence. The platform has many objectives that we will list below. The project started now 3 years ago.



*Figure 27: OpenInterface  
Logo*

The main goal of the OpenInterface project, is to design and develop an open source platform for the rapid development of multimodal interactive systems as a central tool for an iterative user-centred design process. With the objective to integrate any component developed either with Java, C++ or Matlab to any application easily. The platform if therefore turned to the multimodal environment as it process signals and merge or filter them.

Nowadays, OpenInterface has some component such as illustrated in the figure . New components will be integrated after the eNTERFACE 2007 such as head-tracker, OSC Connector component and the ones we are developing in this thesis (pen-based tablet recognition component, hand gestures recognition, String-to-String mapping component). But currently only the components on the figure are available on the OpenInterface Strep web-site. As we can see, there is a Speech Recognition component, a mouse component and two applications ImageReader which enable to put an image on the Byte array and ImageViewer which enable to navigate ( zoom and translate) in an image.



*Figure 28: OpenInterface currents components*

### OpenInterface architecture :

OpenInterface can be seen as a large jigsaw. In the platform, the heterogeneous components are like pieces of a jigsaw that can be registered as reusable and inter-connectible components. Each heterogeneous component is described in XML according to the CIDL – Components Interface Description Language. Each heterogeneous component is encapsulated within a C++ proxy and registered as a plug-in. Components are retrieved through the Graphic Editor by the user who can then edit the components properties and compose the execution pipeline of a multimodal application (in fact this graphical editor is not yet finished). This execution pipeline is sent interpreted by the OpenInterface Kernel (C/C++) to run the application. While designing multimodal applications, the designer needs to specify the multimodal interaction dedicated to a given task of the interactive system under development. To handle multi-modal inputs/outputs issues OpenInterface integrates the concepts proposed in ICARE platform. Two kinds of components are considered: (1) elementary components that enable the designer to define “pure interaction modality” and (2) generic composition components that enable the designer to specify combined usage of modalities.

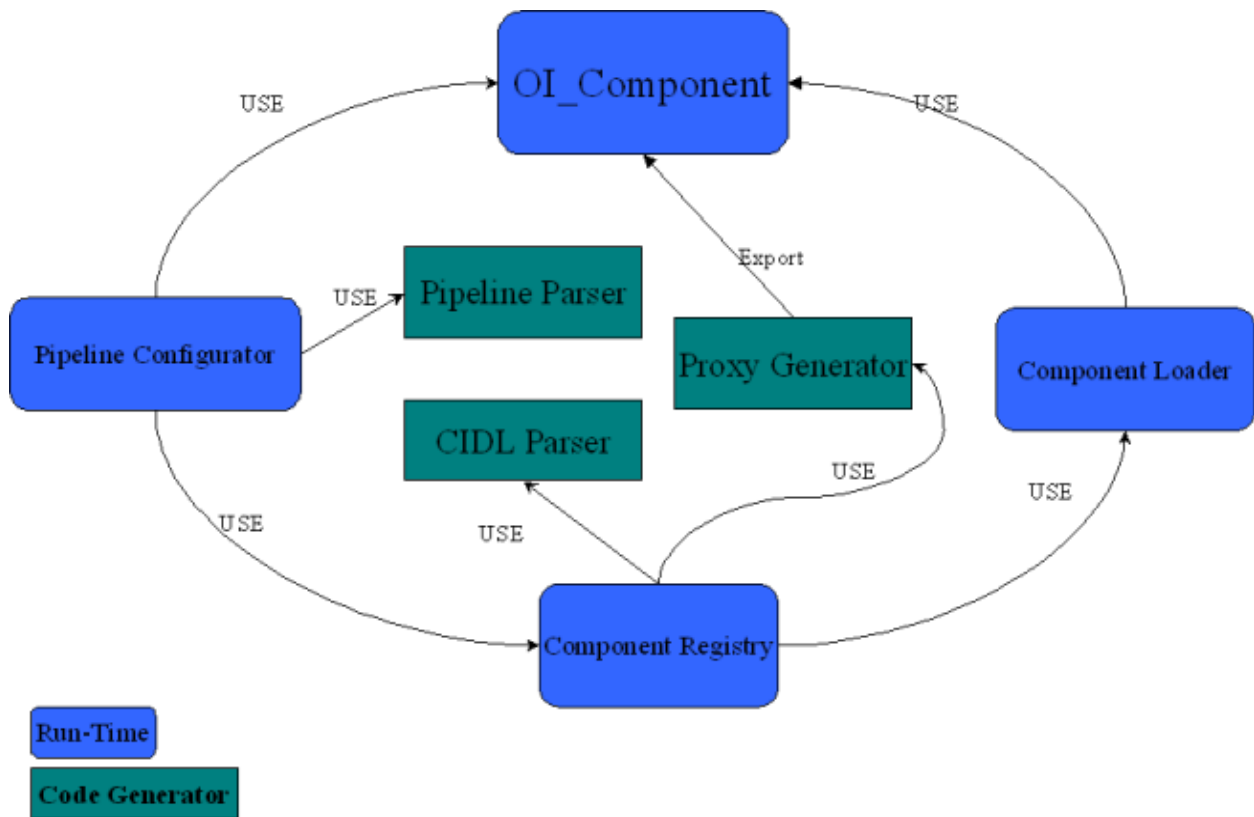


Figure 29: OpenInterface architecture

OpenInterface is therefore based on 2 main concepts :

**components:** A bundled piece of software that provides a set of services/functionalities. The provided software can do anything, ranging from input devices driver, signal-treatment algorithm, network module, graphical interface, etc. A public repository of components has been set up to centralize all existing software and enforce easy reuse of components written by other people.

pipeline of components: It is the interconnection schema of a set of components. It describes and set up the communication channel between the components. An application is then described by a pipeline interconnecting a set of components.

### **General Principle :**

The OpenInterface platform is then connecting different components. How do the platform connects the application and interacting components is the tricky thing about OpenInterface. Let us explain how components are integrated to the platform. Each new components has to have its own CIDL (Component Interface Description Language) which is an .xml file describing the component. In this file, there's nested mark-up which define different stuffs. After describing the components and the CIDL description of the application, components and application are connected into another XML file called pipeline. Pipelines are also an .xml file called PDCL (Pipeline Description and Configuration Language) which define how components and application (which is a component-like) are communicating. We only listed here the main mark-ups, but you can have the complete definition of CIDL and PDCL on [LAWS 06].

#### ***a) The component CIDL description:***

The CIDL has for objectives to define the component in terms of instantiation and communication interfaces. The most important mark-up of this file are (the complete CIDL of our components are given in annexes) :

- <Component> is the first mark-up of the file. All others mark-ups are nested in Component.
- <Container> inside this mark-up, we define the name, the location, and the programming language of the component.
- Then, comes the Mark-up which encompasses the most important Mark-ups of the file, <Facet>. The Facet element is used to describe logical unit inside a component. It will provide a description of the binaries through <Bin>. The <Factory> Mark-up defines how components are initialised. The <Sink> and <Source> Markup which enable the component to communicate with other components.
- <Factory> is important because as we said, it's where the component is instantiated. The interface (i.e. The constructor or factory function to call) of the factory and the format of the created facets must be known. The <interface> gives, the signature and return type of a factory function or constructor to call for the creation of the facet.
- <Sink> and <Source> represent ways for components to exchange data. A sink pin is a way for the component to receive data and source pin a way to send them. Something specific about source pin, is the callback setter attribute. A callback is the way for a component to send asynchronous events. That is similar to call a function the component doesn't know at compilation time. For instance, a mouse driver component would expose a callback for notifying about the current mouse position and buttons states. A callback setter will then be called to register the external function. So instead of polling the mouse component for its state, the user will be notified by the registered

function only when the state has changed. So it means, that once the event is in the source, the source knows only at execution time where it have to send the result and the setter gives the opportunity to others component to register to those events.

#### ***b) The pipeline PDCL description:***

As we said before, the pipeline file is also an .xml file which define how components (and applications which are also considered as components) are communicating. This communication is done by plug in the different source and sink between each others. As the component, we will explain the main Mark-ups of the PDCL file :

- <ComponentList> in this mark-up, all the component that will be used are listed.
- Then <FacetList> where for each component we list the facet that will be used. Notice that some component need to be initialised with a parameter, you define here a Factory mark-up.
- Then come the most importants Mark-up of the pipeline file. <PinList> we will list here the sinks and sources we defined in the CIDL file of each component. It's therefore the list of the communication interface.
- After defining those Pins, we introduce the <Pipe> mark-up which in fact is the «connection manager» for each communication from a component to another, you declare a <plug source= « x » target= « y »>. This means that data will go from x to y where x and y are the pins defined above. You also have specified which argument you want to send and receive according to the others components CIDL.

So we see it's quiet easy to connect components. With a little practice, programmers should be able to develop and connect components.

Furthermore, OpenInterface Strep is developing others tools which should make integration of components for OpenInterface easier. In fact, two projects are in development. The first aim to auto-generate the pipeline between any number of components you like and the second is aiming at auto-generating the CIDL of a component. With those 2 comings tools, OpenInterface will be an easy tool for integrating any modality.

#### **Similar project**

OpenInterface is a part of the Similar project. Similar, a program support by the European commission, describe itself as : « *The European task force creating human-machine interface SIMILAR to human-human communication* ». [NetLink10] Similar works on research about multimodal interfaces to provide computer capacity to communicate as a human with different kind of modality such as speech, gestures, vision, haptics and direct brain connections modalities.

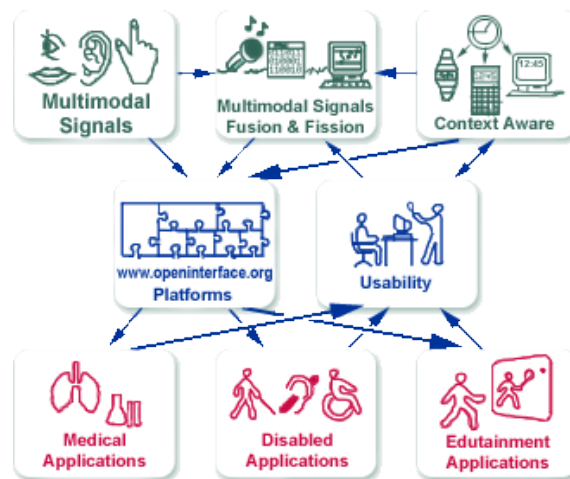


***Figure 30: Similar logo***



As presented on the similar website the european project Similar vision are:

- SIMILAR will create an integrated task force on multi-modal interfaces that respond efficiently to speech, gestures, vision, haptics and direct brain connections by merging into a single research group excellent European laboratories in Human-Computer Interaction (HCI) and Signal Processing.
- SIMILAR will develop a common theoretical framework for fusion and fission of multi-modal information using the most advanced Signal Processing tools constrained by Human Computer Interaction rules.
- SIMILAR will develop a network of usability test facilities and establish an assessment methodology.
- SIMILAR will develop a common distributed software platform available for researchers and the public at large through [www.openinterface.org](http://www.openinterface.org).
- SIMILAR will establish a scientific foundation which will manage an International Journal, Special Sessions in existing conferences, organise summer schools, interact with key European industrial partners and promote new research activities at the European level.
- SIMILAR will address a series of great challenges in the field of edutainment, interfaces for disabled people and interfaces for medical applications. Natural immersive interfaces for education purposes and interfaces for environments where the user is unable to use his hands and a keyboard (like Surgical Operation Rooms, or cars) will be dealt with a stronger focus.



*Figure 31: Similar context*

### eNTERFACE workshop

Thanks to our supervisor, Jean Vanderdonckt, and the Similar network we had the opportunity to participate to eNTERFACE workshop 07 at the Bogaziçi University in Istanbul from 4<sup>th</sup> to 11<sup>th</sup> of Augustus. During this week we participated on the OpenInterface project and worked with Lionel Lawson and Marcos Serrano on the integration of our both modalities to the OpenInterface platform.

*« The eINTERFACE summer workshops, organized by the SIMILAR European Network of Excellence, are a new type of European workshops. They aim at establishing a tradition of collaborative, localized research and development work by gathering, in a single place, a group of senior project leaders, researchers, and (undergraduate) students, working together on a pre-specified list of challenges, for 4 weeks. Participants are organized in teams, attached to specific projects related to multi-modal interfaces, working on free software. eINTERFACE'05 was held at Faculté Polytechnique de Mons, Belgium, in July-August 2005. The eINTERFACE'06 workshop will be organized in Dubrovnik, Croatia, in July-August 2006 ».* [NetLink11]

As you can imagine this workshop were above all for us a invaluable human experience, working with the best European researchers of the multimodality field, working directly with the OpenInterface developer for our specific work and obviously living one week in this large and splendid city of Istanbul along the Bosphorus. This week also permit us to work effectively on our OpenInterface part presented in chapter 5. Nearly all the implementation was made during this workshop and we discover, during discussions, a way to go further in this work as explain later in this paper.



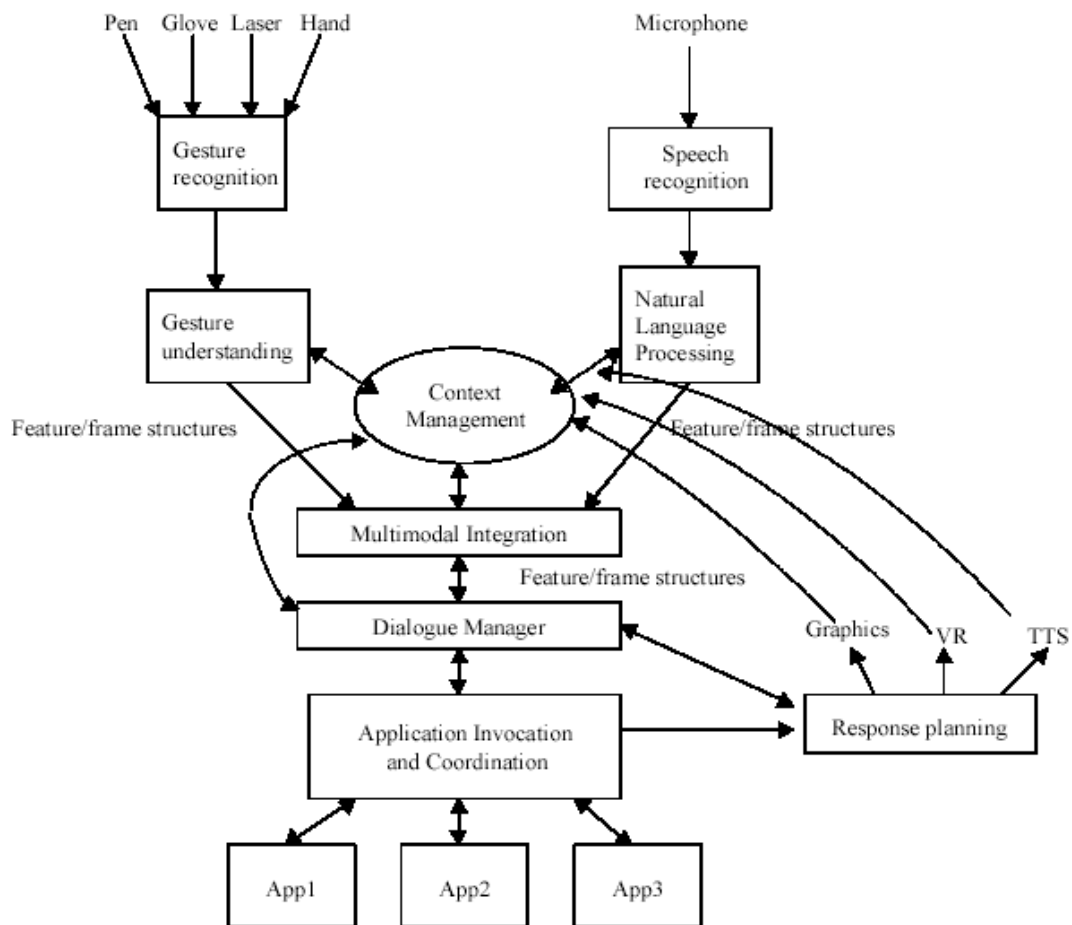
*Figure 32: Bogaziçi university*

### **3. Design of multimodal interfaces**

In this chapter we will explain all the current existing design of multimodal interfaces. We will therefore introduce the most known architecture for achieving multimodal interfaces. As we have introduced the different gestures we can recognize with the help of Quill and HandVu, we will show and justify the choice we made for each gesture corresponding to an action both for pen and hand gestures.

#### **3.1 Multimodal architectures**

Multimodal architectures and more generally interactive systems architecture differs a few from classical architectures. Here on figure 33 is an architecture for processing pointing device and speech recognition. This architecture is quiet easy to understand except context management and multimodal integration which are specific to multimodal architecture.



*Figure 33: Typical information processing flow in a multimodal architecture designed for speech and gesture.*

Figure 33 illustrates two input modes (e.g., speech and manual or pen-based gestures) recognized in parallel and processed by an understanding component. The results involve partial meaning representations that are fused by the multi-modal integration component, which also is influenced by the system's dialogue management and interpretation of current context. During the integration process, alternative lexical candidates for the final multi-modal interpretation are ranked according to their probability estimates on an n-best list. The best-ranked multimodal interpretation then is sent to the application invocation and control component, which transforms this information into a series of commands to one or more back-end application systems.[OVIA 02]

#### a) MVC architecture :

MVC is the most known architecture. M stands for model, V for view and C for controller. Model represents the behaviour of the application, data-processing etc... The view renders the model into a form suitable for interaction, typically a user interface element. And the controller processes and responds to events, typically user actions, and may invoke changes on the model.

Typically, it works as follow :

The user interacts with the user interface in some way (e.g., presses a button).

1. A controller handles the input event from the user interface, often via a registered handler or callback.
2. The controller accesses the model, possibly updating it in a way appropriate to the user's action (e.g. : controller updates user's shopping cart).
3. A view uses the model to generate an appropriate user interface (e.g. : the view produces a screen listing the shopping cart contents). The view gets its own data from the model. The model has no direct knowledge of the view.
4. The user interface waits for further user interactions, which begins the cycle anew.

However, MVC architecture have drawbacks, the controller and view are not completely independent of the data representations. Furthermore, there's a direct connection between model and view, that the controller don't see. That's why MVC doesn't really fit to multimodal or generally speaking interactive systems. We need another architecture where interaction components are more independent. Let's have a look to the Arch model.

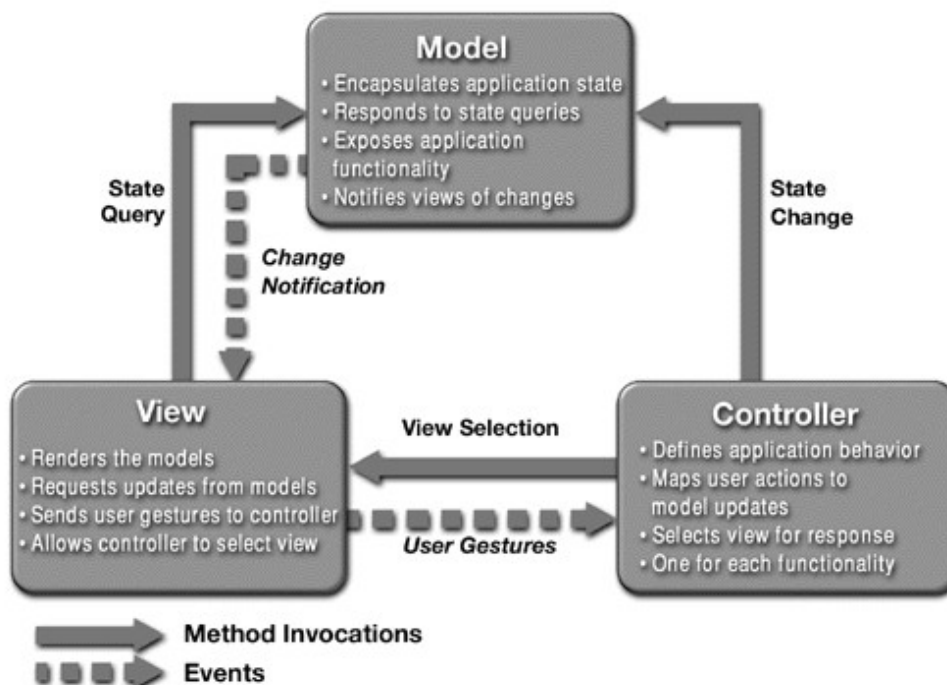


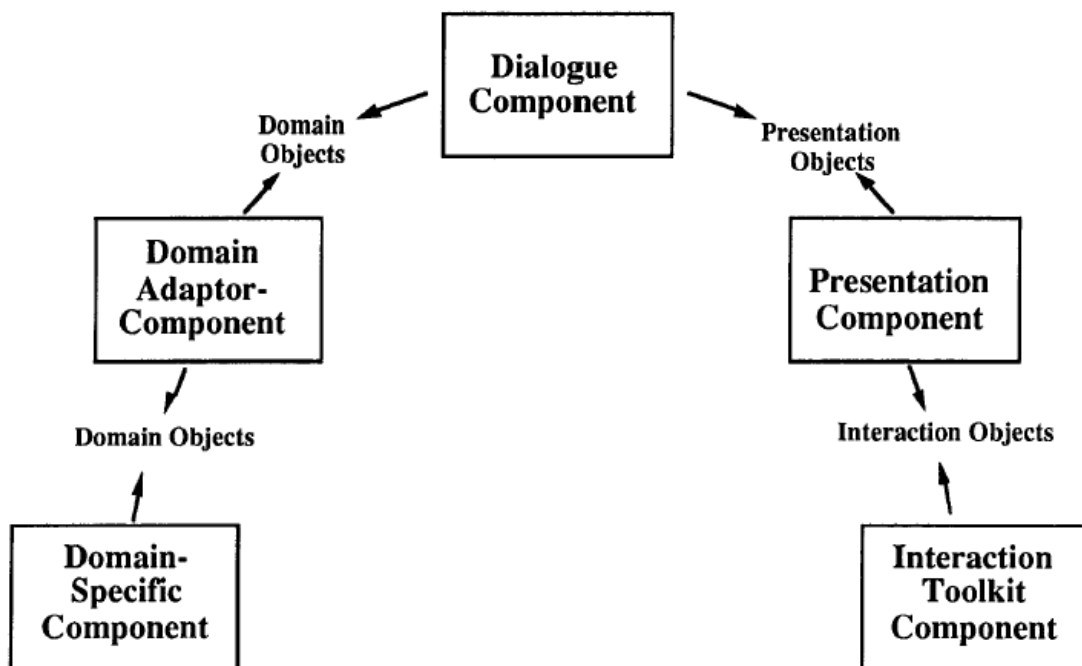
Figure 34: MVC architecture

**b) ARCH architecture:**

The idea of the ARCH architecture, is that when developing the architecture of the system, engineers decide the criteria they want to met such as system runtime performance or any other criteria they are judging useful. In the paper of the UIMS [UIMS 92], they show that one architecture is impossible for meeting all the criterions at a same time. So the designers have to make trade-off between for example the criteria of not suffering from the effect of a changing technology and improving the system runtime performance.

ARCH architecture insist on the minimizing effort due to changing technology. (e.g. : buffering the remainder of the system from the effects of evolving Interaction Toolkits). As we can see on figure 35, this architecture looks like an arch. The advantage of such an architecture, is that we define an architecture which minimize the future effects of changing technology. That is if new modal devices are created the ARCH architecture should give us the opportunity to integrate them easily.

For example Dialogue-oriented that is system that have extensive capabilities for mapping user actions into the behaviour of the interface - managing windows, controlling appearance, choosing different techniques for representing the same information, etc. have an arch more oriented on dialogue and presentation component See on figure 36.



*Figure 35: The ARCH model*



Figure 36: ARCH architecture for dialogue oriented systems

ARCH model give then more independence to interaction component by providing more effort in developing independent component to the dialogue.

## 3.2 Pen-based gestures

In the previous chapter we defined the qualities of a good gesture. We focused on the iconic quality, as we could design our gestures thanks to Quill. But also thanks Quill we had an immediate feedback about the gesture similarity for human perception and for the gesture recognizer. We tried to trade-off between both in order to get the satisfaction of both.

Remark that the point on the gestures images are representing the beginning of the gestures.

### Windows managing actions

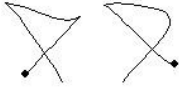








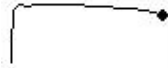
Action	Gesture	Justification
Close		2 gestures are possible for this action. This is representing the cross on the right corner of most of the commons interfaces. We privileged the iconicity
Minimization/ Maximization		Same justification as above
Reduction		Same justification as above

Figure 37: Windows managing actions pen gestures


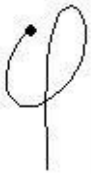
**Browsability actions**

Action	Gesture	Justification
Next item		It's a metaphor for going to the futur to what's coming next. The arrow is pointing to the right.
Previous item		Same justification as above
Up		The gesture starts from the bottom to the head.
Down		Same justification as above
Left		Same justification as above
Right		Same justification as above
Back		This gesture is maybe less iconic but we had to find a gesture that mean the same as the return key and as left gesture was already token we choose this.

*Figure 38: Browsability actions pen gestures*















**Validation actions**

Action	Gesture	Justification
Selection		The V of victory is often use as a validation action. It's almost the same as OK.
Reinitialisation		It's maybe the most complicated gesture for the commands actions. But it reminds the « phi » greek letters meaning nothing. This action is resetting the content of a text field. Setting it as default.

*Figure 39: Validation actions pen gestures***Characters gestures.**









Concerning the choice of the Characters and numbers gestures, we inspired us from Graffiti, then we customized some letters in order that Quill recognize them better. Some letters are more iconic than others but we had to make a trade-off between effective recognition and learnability.

<i>Letter</i>	<i>Gesture</i>	<i>Letter</i>	<i>Gesture</i>	<i>Letter</i>	<i>Gesture</i>						
A		B		C		D		E		F	
G		H		I		J		K		L	

M	N	O	P	Q	R
S	T	U	V	W	X
Y	Z	space			

Figure 40: Characters pen gestures

#### Number Gestures :

Number	Gesture	Number	Gesture
0		2	
1		3	
4		6	
5		7	

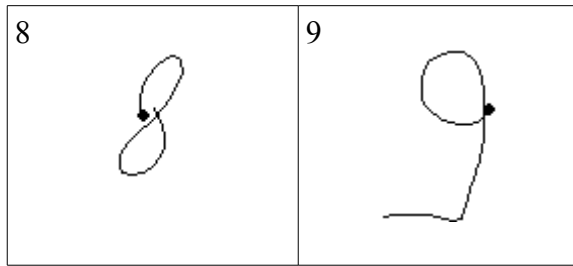


Figure 41: Numbers pen gestures

### 3.3 Hand gestures

As explain in section 2.6 the hand gesture recognition software we choose for implementation can only recognize six different gestures. More gestures are currently in development by a developer's community. We thus present here hand gestures associated to each action even if we know that it won't be possible to entirely implement them.

**Comment :** As for pen gesture, here something with a right direction symbolize a forward movement, a progression and on the opposite with a left gesture a return or a backward movement.

#### Windows managing actions







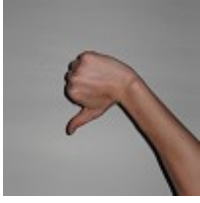
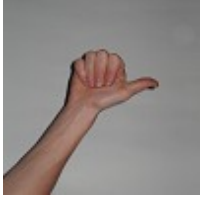
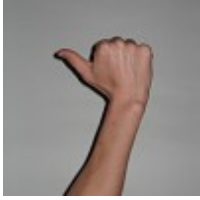
Action	Gesture	Justification
Close		A close hand for a close action.
Reduction		Symbolise take the windows whit the little finger (auricular) and put it down in the task bar symbolize by the thumb
Minimization/ Maximization		Victory gesture can represent the action to realize something great, so to put maximize



Figure 42: Windows managing hand gestures

**Browsability actions**

Action	Gesture	Justification
Next item		The index finger up and the thumb on the right to indicate the direction to continue something, so to pass to next ones
Previous item		The same but with the thumb left to indicate the return, the previous ones
Up		The hand close and the thumb open indicate the a direction, so here up
Down		Here bottom
Right		Here right
Left		And here left

*Figure 43: Browsability actions hand gestures*

**Validations actions**

Action	Gesture	Justification
Selection		An open hand in front of the webcam symbolize a strong action as a selection, a click on a button, ...
Reset		A shaped hand symbolize with the fact that's open on left, a return also. But here a reset.

*Figure 44: Validation action hand gestures*

Each gestures is relatively easy to do, except this one for reduction which need a few training. Gestures for next and previous item are sometimes associate with a zoom actions. But there here associated with browsability actions because there are very complementary. We need only to return the hand to go from next to previous item.

## 4. InterpiXML Development

Because our development is based on 2 modalities (hand and pen) and on a combination of those 2 modalities but also based on 2 platforms (OpenInterface and InterpiXML) we will expose in this two next chapters the development of each modality in each environment.

We can represent that like this (Numbers are chapters numbers) :

<u>Environment</u>			
OpenInterface	5.1	5.2	5.3
InterpiXML	4.1	4.2	4.3
	Hand	Pen	Hand & pen
	<u>Modality</u>		

*Figure 45: Reading plan for implementation chapters*

But before explaining directly how we added hand recognition on InterpiXML, we will define some change we had to perform on the platform.

### From InterpiXML v1.0 to v1.1

InterpiXML v1.1 is different from previous version in two things. First because it has been upgraded to version 1.8.0 of UsiXML language. This new version permit to specify multimodal interfaces. Second because it add concept of multimodality interaction to interfaces with a new architecture to easily adapt new modalities.

## Adaptation to UsiXML v1.8.0

Before beginning to add multimodality to InterpiXML we need first to adapt it from UsiXML language version 1.6.4 to version 1.8.0. To permit to specify multimodal interfaces as say on the website. «*The UsiXML language is currently evolving in order to encompass full multi-modal user interfaces so that they are compliant with the tools produced by the OpenInterface platform and to go beyond multi-modal web user interfaces which have been addressed so far*». [NetLink12]

We present here principal changes with previous version. To be conform with new version we used to :

- add new Layout as **BorderLayout**, **GridLayout** and **FlowLayout** which were not described in previous language version.
- replace previous **TextComponent** to new objects described in new language version : **InputText** and **OutputText**.
- add the possibility to insert images in the interfaces. Images as text can be relative to a ContextModel and then can be different according to the language choose for rendering. Images can have :
  - relative adresses (« *Tests\LogosGoogle\logoIt.gif* »)
  - absolute addresses (« *C:\LogosGoogle\logoIt.gif* »)
  - or url addresses (« *http://www.google.it/intl/it\_it/images/logo.gif* »)

and other possibilities as to choose a background color for each component, ...

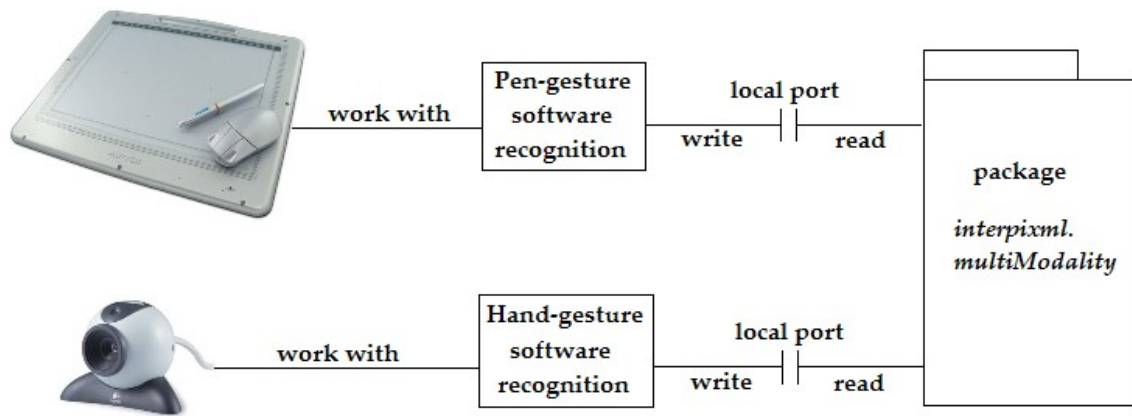
## Adaptation to multimodality

We will first explain the adaptation of InterpiXML architecture to accept any kind of new interaction modality. At the beginning, InterpiXML produced interfaces which support only keyboard and mouse interaction, as any other interfaces.

The approach is that any modality which want to interact with interface can do this by one local port. Modality write on a local port what it wants to communicate to InterpiXML. Then InterpiXML examine message and transmit it by posting it on a event bus. All interfaces read what is posted on this bus. If several interface are open, only this which get the focus interpret and react to this message.

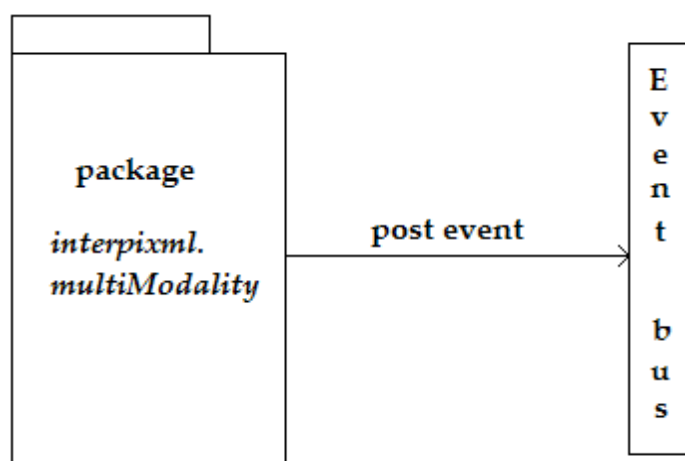
So, we added a new component in InterpiXML which is a kind of middleware between the modality and the event bus. The goal of this package is to provide an easy way for InterpiXML to read messages on local port and write them on event bus.

We will here explain the four steps used to communicate from an event (e.g. : Gesture detection) detected by the modality to the action done on the interface.

**1st step :**

*Figure 46: First step in the modality – interfaces communication*

Each recognition software work with his own modality and detect gesture from this one. InterpiXML is not in charge of this part. The recognition software must be adapted to write messages on a specified local port. Messages are probably composed of gestures recognized. In this first step package *InterpiXML.multiModality* is here to read messages on this different ports. The package is in charge of several other things in the second step.

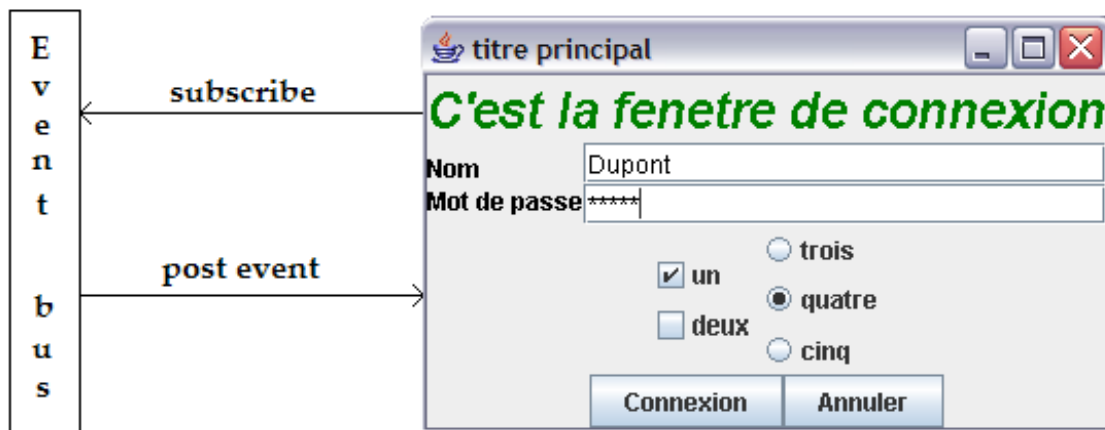
**2nd step :**

*Figure 47: Second step in the modality – interfaces*



The second role of the package is to explore the received messages, get the significant informations and transform it into events which will be comprehensive for interfaces. After this the package post this new event on the bus event. Two different messages can come from different modalities but represent the same action for interfaces. So messages from software recognition are independent.

### 3th step :



*Figure 48: Third step in the modality – interfaces communication*

In this third step, the event will be posted to each interfaces which had previously subscribe to receive this event type.

Event bus work like this : Each component can post events on bus (here only classes from *InterpiXML.multiModality* package do it). And then, only components which have subscribed to receive specific type of event will indeed received them. In our case at the construction, each interfaces subscribe to all events known by InterpiXML.

### 4th step :

When one interface received one event, it first ask if it get the focus. If not, it do nothing. If it's case, it will react to the event depending on the event type.

## Architecture

To better understand the new InterpiXML architecture and how these components works together we will present the new classes added to the interpreter for the multi-modality and a typical sequence diagram. First we show the ARCH architecture we adapted to InterpiXML according to the definitions given in the third section and a levels-based architecture which is not very formal but permit to better see connections and relations between each components.

### Arch architecture :

As we discussed on chapter 3, the ARCH architecture insists on dialogue and presentation. It means the shifting between the input mode (webcam, tablet) and the interpretation of the commands. As we can see inserting a new modality in this architecture is quiet easy since we only have to integrate another device reader into the presentation layer. However this architecture don't support fusion of data for achieving this, we had to add a layer called multimodal fusion for example after the device reader and realise this fusion. How we could implement this fusion is discussed in section 4.4.

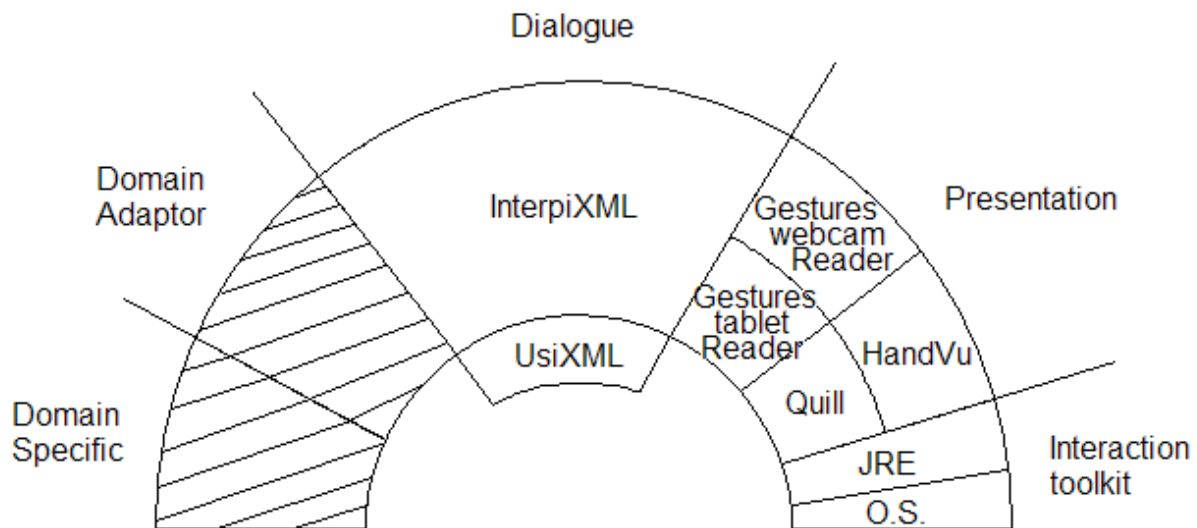


Figure 49: InterpiXML ARCH architecture

## Levels-based architecture

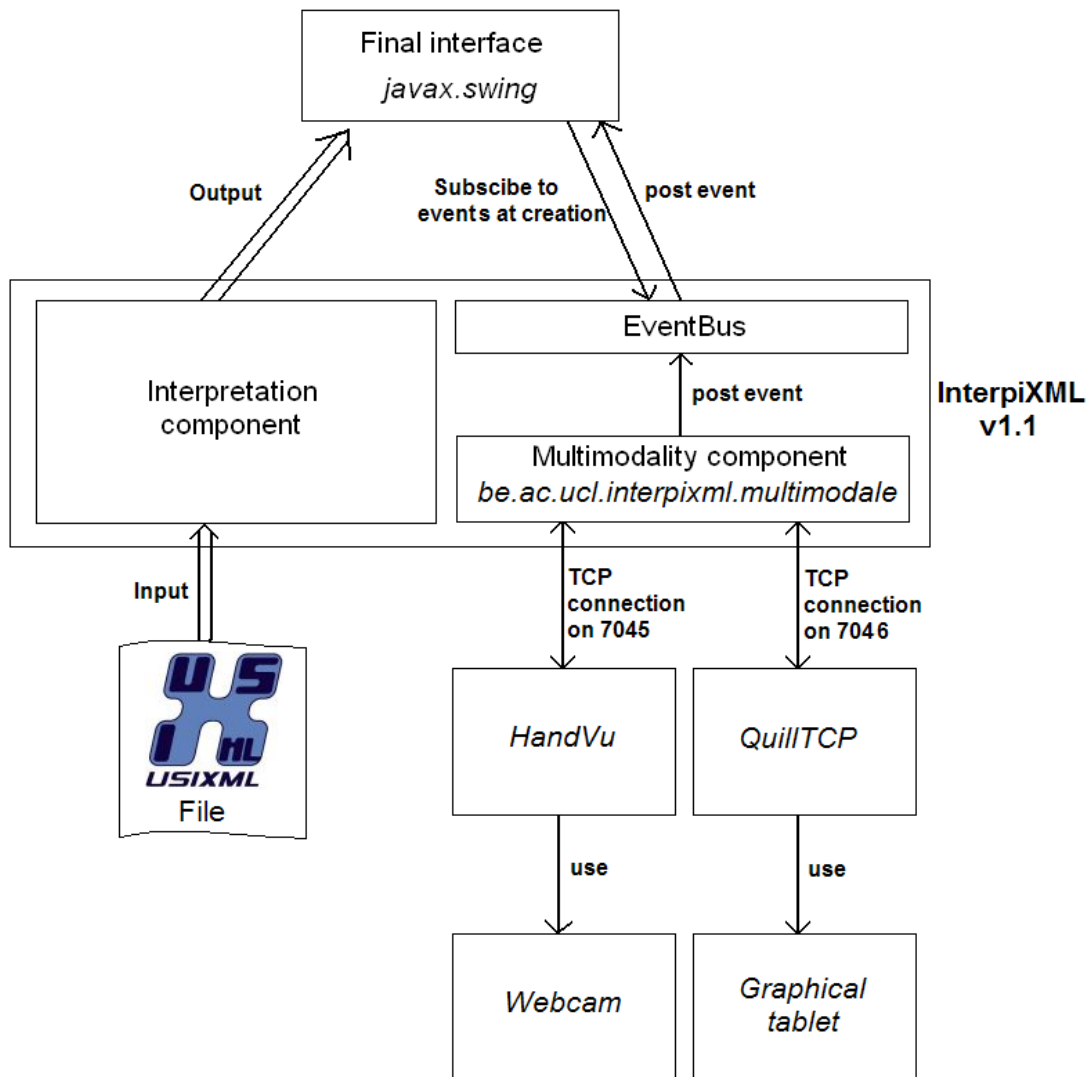


Figure 50: InterpiXML architecture

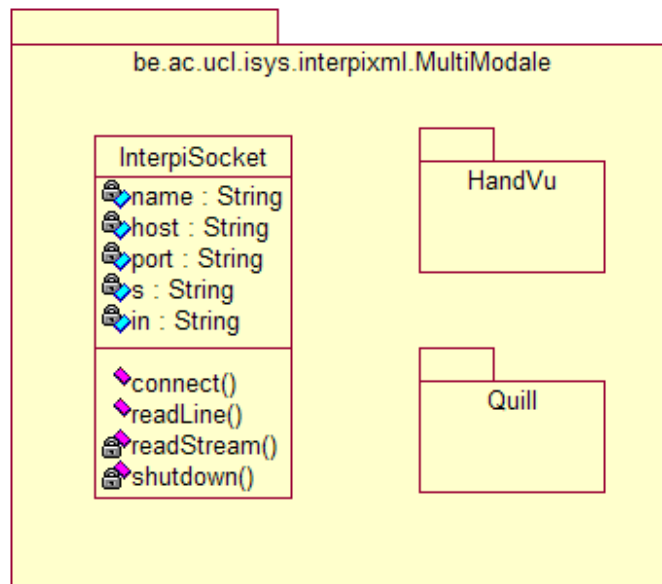
This need some explanations. If we begin with the bottom that we call level 0 : We represent the hardware components, the webcam and the graphical tablet. We decide to not put all the hardware components (computer, screen, ...) to simplify the figure. Then the two hardware components are used by their respective software recognizer (*HandVu* and *Quill*) at levels 1 which contains all external resources to InterpiXML and where we decide to integrate also the *UsiXML* files.

If we decide here to cut vertically the figure in two. You can see on the left the traditional InterpiXML v1.0 architecture which get as input a *UsiXML* file, interpret it and produce as output a Java swing interface.

The more interesting, in case of this thesis, stand on the right. The two software recognitions communicate with local TCP connection on port 7045 and 7046 with the new specific InterpiXML component (locate in package *be.ac.ucl.isys.InterpiXML.multiModale*) on level 2 and stand inside InterpiXML v1.1. This component check messages received from levels 1 and if messages represent actions for interfaces, it post event corresponding to this actions on the eventBus. The event can be *HandVuEvent* event or *QuillEvent* event. Now the eventBus need only to send this event to all interfaces which have subscribed at their creation to received this kind of event. Finally only interface which get the focus react to this event by the associate action.

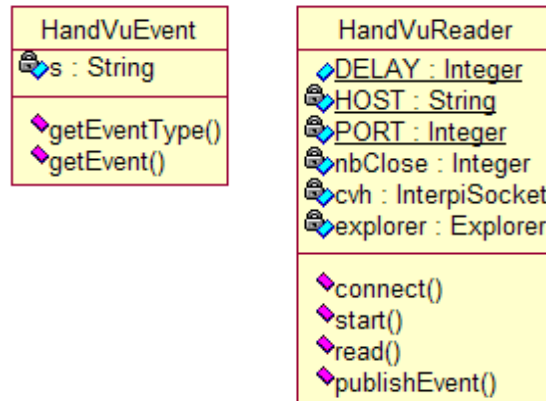
### Class diagram

Here is the principal package add to the new InterpiXML version to take into account the multimodality. There are a class *InterpiSocket* which provides services to connect and read messages on a specific port and two package *HandVu* and *Quill* which will be explain later.

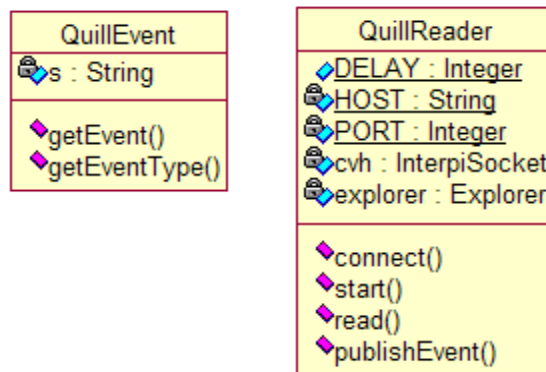


**Figure 51: Package**  
*be.ac.ucl.isys.InterpiXML.multiModale*

Here is the *HandVu* package with its class *HandVuReader* which provides services to analyse messages which come from its *InterpiSocket* on local port 7045. And if this message contains interesting gestures for interfaces, it post an *HandVuEvent* on the eventBus. The package *Quill* provides the same services but with different event post follow the messages received on local port 7046.



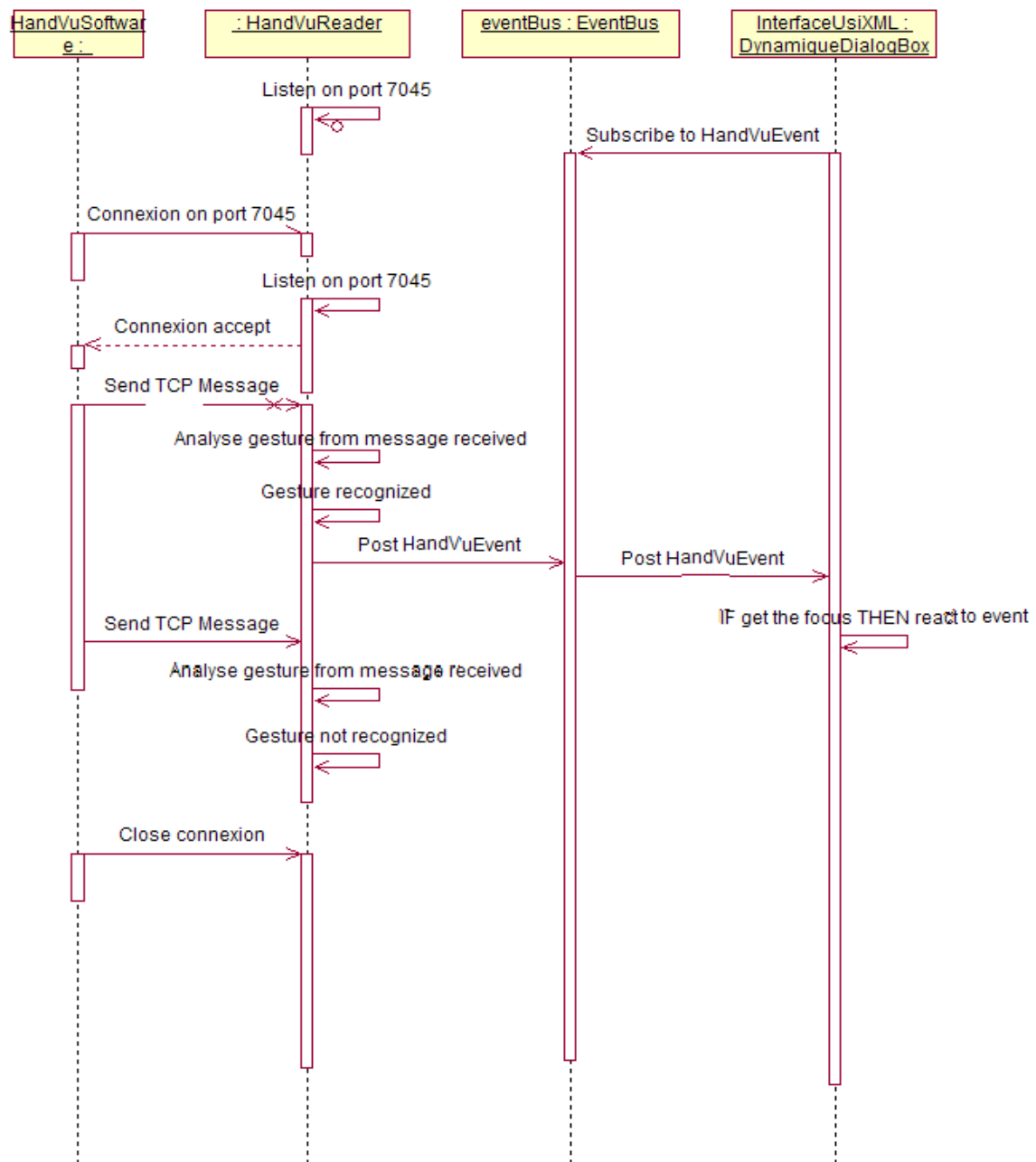
*Figure 52: handVu package*



*Figure 53: quill package*

### Sequence diagram

On figure 54 you can find a typical running example. InterpiXML, and the HandVuReader starts first and HandVu software later, but it can be in the other way. Every interface interpreted subscribe to the eventBus to received HandVuEvent. Then when the connexion is ok between HandVu software and the HandVuReader (or InterpiXML) messages can ben received by the interpreter. When a gesture is recognized a corresponding HandVuEvent is posted on the eventBus which send it to all interfaces which have subscribed. If no gesture recognized in the message HandVuReader do nothing and wait for next message. Finally, there is no importance if that is the HandVu software or InterpiXML which decide to stop its execution first. The connexion just stop and one of the two components (HandVu or InterpiXML) can continue to work without the other. For Quill this is just the same but with QuillReader and QuillEvent post on the bus.



*Figure 54: Typical execution between HandVu software and interfaces produce by InterpiXML*

This architecture permit to easily add or retrieve modality in the future. The addition can be very easy. Three things must be modified :

1. Modify the software recognition modality to write its messages on specific port.
2. Add a new class in the package *InterpiXML.multiModality* which manage connection between the modality and the event bus.
3. And then modify the interfaces to subscribe to event from this new modality and to interpret and react correctly to the message received from the event bus.

## 4.1 InterpiXML and hand-based recognition

### 4.1.1 Architecture

As show in previous section in the architecture presentation, we then use the *InterpiXML.multiModality* package to add our new component.

### 4.1.2 Implementation

For HandVu, no much change were necessary. When you install and run HandVu beta 3 on a windows platform, it already write its results on a local port (**7045**). However under Linux its not case, so it need to download special version with the *GestureServer* added to write on this port. And then you can run it normally before or after that InterpiXML started. Gesture will be recognized by the software as usually and results written on local port **7045**. You can verify that with a telnet on this port and then see messages that HandVu write.

It's now to InterpiXML to read this messages and send it to interfaces. All this problematic will be contained in the new specific package.

When InterpiXML is starting, it also start a thread to instantiate the class *HandVuReader*. This object, from class locate in *InterpiXML.MultiModale.HandVu* package, try to connect to local port 7045 every ten seconds. When connection established it begin to read message from this port, convert them in specific format for interfaces and then post it on the event bus.

This specific format encapsulated in a *HandVuEvent* class. It compose of two parts separate by a colon. First part described the message type and the second part represent the message itself.

We present in next tab different gestures recognition which can come from HandVu messages, their *HandVuEvent* messages associated posted on event bus and their actions associated.

HandVu gesture name	Message posted	Action
open	Command:Select	Selection
Lpalm	Command:Next	Next item
Lback	Command:Previous	Previous item
closed	Command:Close	Close
victory	Command:MiniMaxi	Minimization/ Maximization
sidepoint	Command:Reduce	Reduction

*Figure 55: Gesture name - message posted - action associations*

Because the close command is relatively destructive, it post only if it read (recognized) three consecutive times. It's now to interfaces to receive these messages a react to them. During the interfaces construction, each of them subscribe to receive *HandVuEvent* event. Then, all messages posted on event bus will call the *onEvent(Object e)* of each interface. They only need now to react correctly to each event received by this method. Here is a sample of this method :

```

public void onEvent(Object e) {
    String type="", evt="", from="";
    // If HandVuEvent
    if(e instanceof HandVuEvent) {
        HandVuEvent hvEvent = (HandVuEvent)e;
        type = hvEvent.getEventType();
        evt = hvEvent.getEvent();
        from="HandVu";
    }
    // Si la fenetre contient le focus
    if(this.isFocused()){
        // Si c'est une commande
        if(type.compareTo("Command") == 0) {
            // Close
            if(evt.compareTo("Close") == 0) {
                this.dispose();
            }
            // Next item
            else if(this.getFocusOwner() != null &
                evt.compareTo("Next") == 0) {
                this.getFocusOwner().transferFocus();
            }
            ...
        }
    }
}

```

*Code 1: Interface reaction to event implementation*



First, interfaces cast the received event to see if it's really a HandVuEvent and then get the type and the event itself. Interfaces react to event only if it's the focus owner and then process normally the message.

### 4.1.3 Examples

Here is a typical example of a utilisation case with one interface open. InterpiXML on the left have a label to inform the user about HandVu connection. On the right there is the HandVu software. The green rectangle inform that a gesture is recognized. On the interface that work on is this one which get the focus on the bottom.

You can go from item to item, select radio button or check box, click on button and also close, reduce or minimize/maximize the windows only from the webcam. And all that without change in the UsiXML source files.

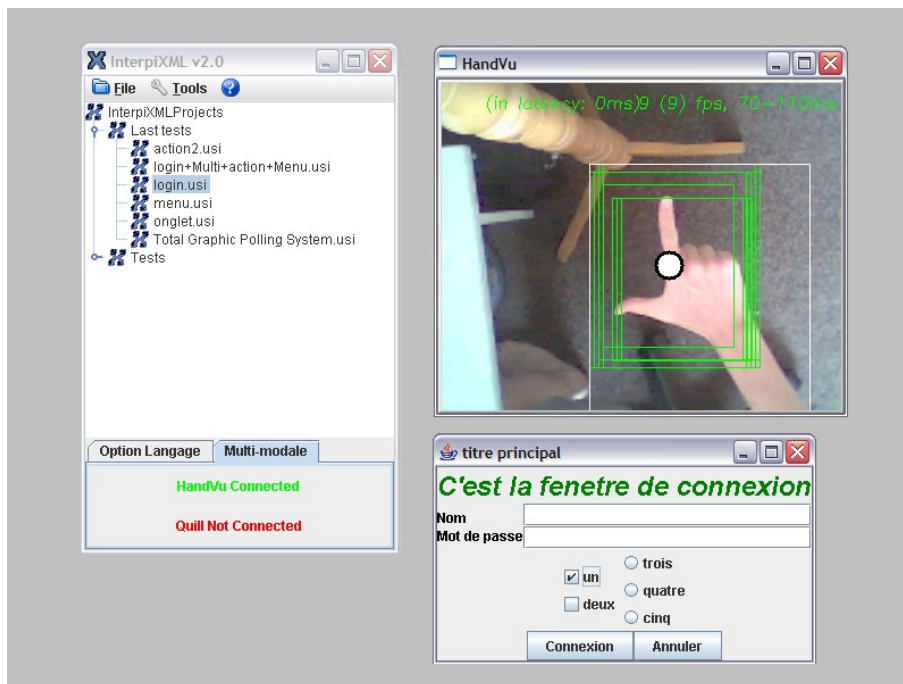


Figure 56: InterpiXML with hand recognition modality

### 4.1.4 Evaluation

Except the sidepoint gesture which is not good recognized by HandVu, all other gestures works fine. With a few training it's relatively easy to work like this. The other problem is that is not possible to enter text in these interfaces. So the interfaces must be specific to work only with this modality, without keyboard and mouse. The last problem is located in HandVu which need sometimes to restart the tracking because it loose the hand (possible to force the reset by pressing the r key).

## 4.2 InterpiXML and pen-based recognition

The pen-based recognition modality implementation is very easy because exactly similar and symmetric to the hand-based recognition modality implementation. The only change is that the software recognition, Quill, can recognize much more gestures and so can provide more different actions to interfaces, that's why we could integrate the numbers and characters gestures.

### 4.2.1 Architecture

Nothing really specific for Quill which like Hand Vu use *InterpiXML.MultiModale* package to communicate with interfaces. We just have to notice here that after defining all the gesture in Quill, we save it in a .gsa file. This file contains features which are essential Quill's recognizer. This file has to be putted in the «Data» directory and this directory has to be put were the .class of the recognizer will be executed. We didn't found any possibility to parametrize the path of this file.

### 4.2.2 Implementation

About the software recognition, we need here to adapt it to write messages (compose of gesture recognized) on a specific port. So we did a second version of Quill names QuillTCP which resolves this problem and write it result on local port **7044**.

As before, a new thread start which instantiate an object from *QuillReader* class at the beginning of InterpiXML which listen every ten seconds on local port **7044** and try to connect to QuillTCP. When a connection established, the object is in charge to read messages and post them on event bus. Here that the same message which come from QuillTCPReader and is posted on event bus under a *QuillEvent* object. In this next tab we present messages received by the *QuillReader* object, the messages that the object post on event bus and the action associated.

QuillReader gesture name	Message posted	Action
Command:Select	Command:Select	Selection
Command:Next	Command:Next	Next item
Command:Previous	Command:Previous	Previous item
Command:Close	Command:Close	Close
Command:MiniMaxi	Command:MiniMaxi	Minimization/ Maximization
Command:Reduce	Command:Reduce	Reduction
Command:Reset	Command:Reset	Reset
Command:Up	Command:Up	Up

Command:Down	Command:Down	Down
Command:Left	Command:Left	Left
Command:Right	Command:Right	Right
Command:Return	Command:Return	Back space
Character:C	Character:C	Insert character C

**Figure 57: Gesture name – Message posted – action associations**

Two last commands are special from Quill and are not declare in possible action on interfaces. From last command, names Character type it's possible to insert character from 'a' to 'z' in and numbers from '0' to '9'.

In order not closing the window by error, we defined for this gesture, as Quill give an estimation of the recognition, a rate of 95%, It means that Quill must recognize this gesture with a rate above 95%. This is because it's a critical action which is irreversible. Something other which is also to notice is that we have two recognizer one for the commands action and the other for characters and and numerical digits. It implies that for designing characters or numerical digit the button of the pen has to be pressed so the QuillModality know which recognizer to call. See on code 2.

```
MultiInterpreter mult=new
DefaultMultiInterpreterImpl();
    Interpreter intrp = new
StandardGestureInterpreter2 (SOCKET, "commandes.gsa");
    intrp.setAcceptLeftButton(true);
    intrp.setAcceptRightButton(false);
    mult.add(intrp);
    Interpreter intrp2 = new
StandardGestureInterpreter2 (SOCKET, "caracteres.gsa")
;
    intrp2.setAcceptLeftButton(false);
    intrp2.setAcceptRightButton(true);
    mult.add(intrp2);
    s.setAddLeftButtonStrokes(false);
    s.setAddRightButtonStrokes(false);
    s.setGestureInterpreter(mult);
```

**Code 2: Code For the command and charcter recognizers.**

Interfaces subscribe to receive *QuillEvent* event and react to each event received if they are the focus owner ones.

### 4.2.3 Examples

The example is very similar to the previous one. A label inform you about the InterpiXML-Quill connection. The Quill panel stand on the right with a selection gesture and just below you can find the interface.

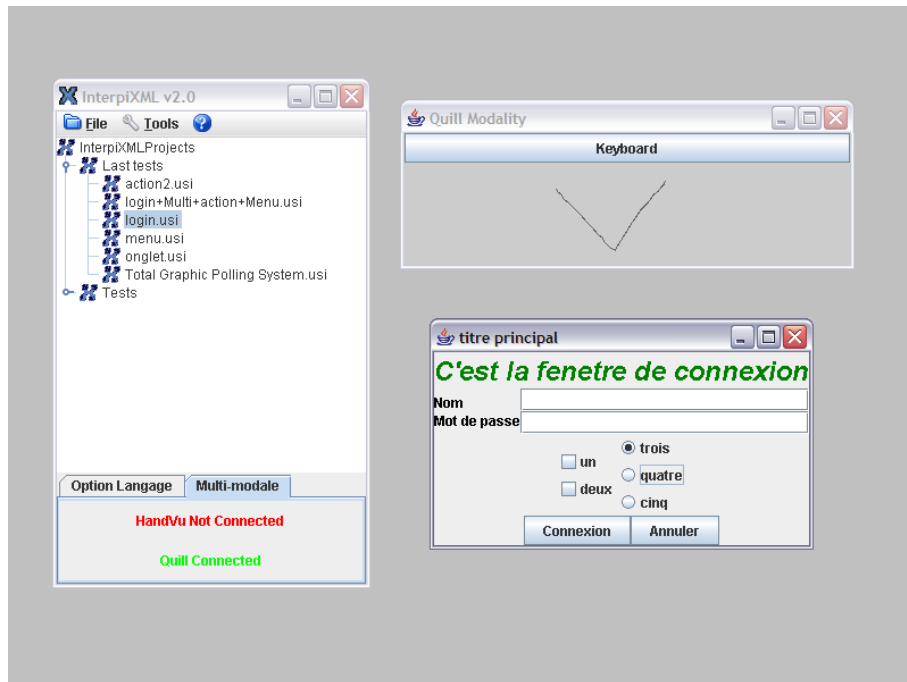


Figure 58: Screenshot pen-based modality and InterpiXML

### 4.2.4 Evaluation

This part of the software is working nicely. Gestures commands are well recognized due to Quill. The misrecognition rate is quiet low. However, for the characters, this misrecognition rate is more high because gesture are more complex for recognizer to recognize. As it was not the purpose of this thesis ( designing good gesture ) we did not spent lot of time on it. But we followed a few the advice given by Quill as we mentioned before. To face the problem of misrecognition rate for characters, we designed a virtual keyboard to help new users in inserting characters. See on figure 59 this keyboard. Note that characters and numbers are drawn by drawing while pushing the button of the stylus.



Figure 59: The QuillModality Keyboard.

### 4.3 InterpiXML with hand and pen-based recognition

Because the two previous modalities are completely independent, the integration of the two ones at the same time is not more difficult to implement. So we won't explain here a new architecture or implementation. You can use no modality, one of two modalities or the two ones in the same times without any change. You just need to start the gesture recognition software and InterpiXML is take in care of the communication between gestures recognizer and interfaces. To add new modality, you just need to do as before for hand or pen gesture modality.

#### 4.3.1 Examples

We present here a example with the two modalities and two interfaces.

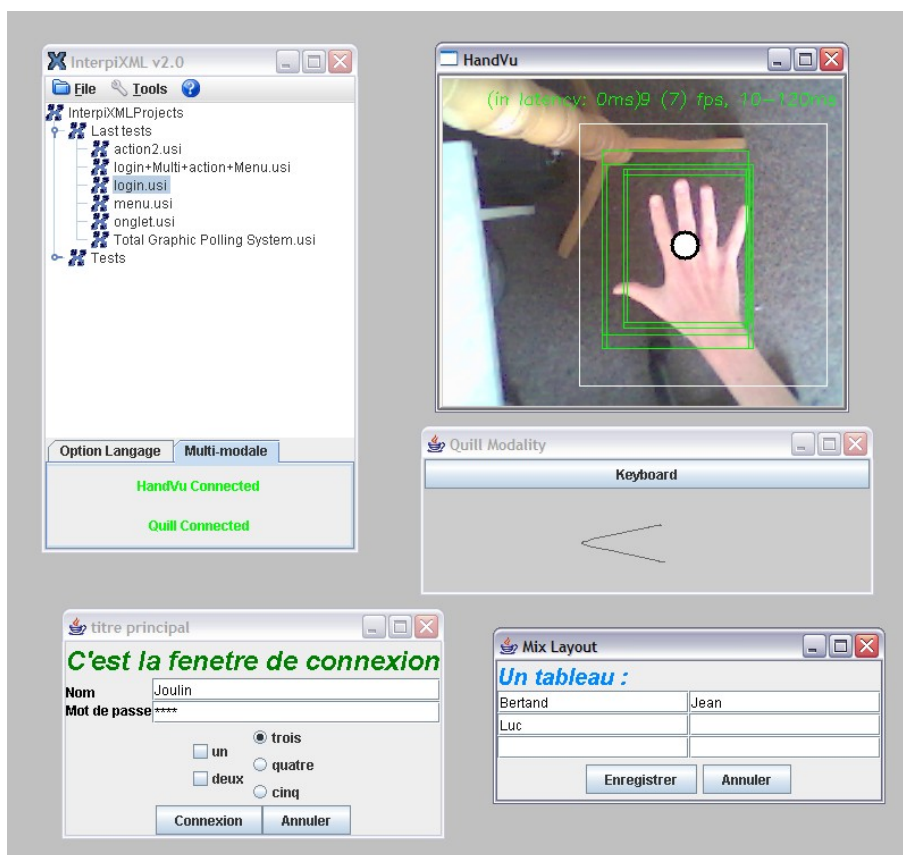


Figure 60: Both modality connected to InterpiXML

#### 4.3.2 Evaluation

The previous example work fine, the only difficulty is to manage with the tablet and the hand in the same time. It's almost impossible to manage executing both modalities at the same time, because you should need your best hand for the two modalities. However it's possible to alternate from modality.

## 4.4 General evaluation

To make a general evaluation of our work concerning InterpiXML, we can first evaluate the new InterpiXML with the CARE properties.

Assigment : Since characters and numbers actions are only possible with the tablet we have here the Assigment property for characters and numbers actions specification. As we said before, some commands are not possible to do with the camera (return, reset) those commands are also Assigment for the pen-based modality.

Equivalence : For all the other commands we can either specify them by pen gesture or hand gesture. The user has the choice we are then talking about equivalence.

Concerning Complementarity and Redundancy, we did not integrated them in InterpiXML for some reasons. First of all our modalities, hand webcam recognition and pen based recognition are not usable for one person at a time. Because the user has to drop the stylus and show his hand or conversely. Secondly, no actions should use complementarity in InterpiXML at the current state of InterpiXML.

However, integrating Complementarity and Redundancy are not so complicated. For Redundancy : a way for doing this is to have a stack on which commands are pushed. To each command is associated a timer determining if we wait for another modality. When another modality command of the same meaning is pushed on the stack the timer could be restarted. When the timer run out of the time, the action is executed and the stack filled out. But this timer has to be very short. In fact, if this timer is too long, user maybe specifying another command and not providing an additional input for the original command.

For Complementary we also would use a stack. If it's a command which needs arguments, we would wait for this argument to come. But, to get exact definition of complementarity, we would check that argument is coming from another modality. When the command gets his arguments, the stack is filled out.

After that we can add more general evaluation about the new multi-modal InterpiXML platform. Strength of this upgraded platform is that the UsiXML file writer which want to realize his interfaces doesn't need to take into account about the modalities which will be plugged at the interpretation time of his UsiXML file and nothing need to be specified in the file. The disadvantage of this new platform is that to add new modality, the developer need to change the core of the platform and be able to change and add new class in. In the next chapter we will work with a platform which prevent this where it's possible to add or delete component and modality easily without need to change core code. Let's see to OpenInterface.

## 5. OpenInterface integration

The second part of our work was to integrate new modality components to the OpenInterface platform. We will present in this chapter how we did it. First how we add the hand gesture recognition (with HandVu), then the pen gestures recognition (with Quill) and finally both on the same application.

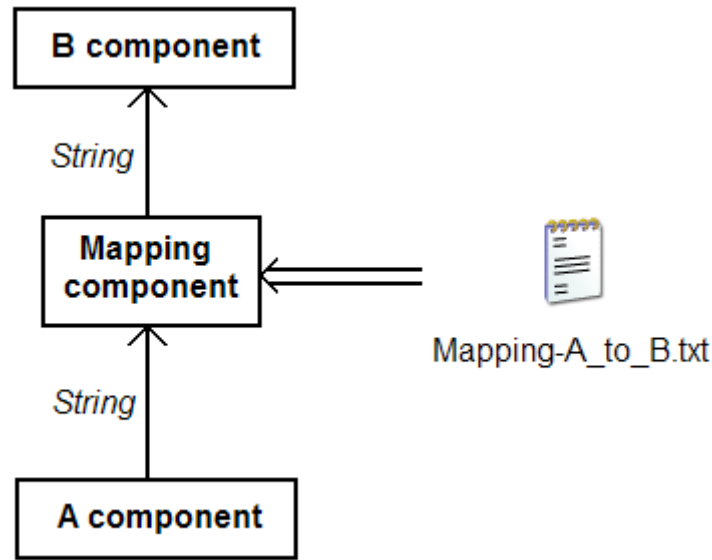
All examples are based on a simple interface : an image viewer. This interface can receive 6 commands : **Next** to see next image, **Previous** to see previous image, **Close** to close the interface, **MiniMaxi**, to put frame in its normal or maximum size, **Reduce** to inconfify the frame and **Modify** to choose if the text field display the received commands or the images path. So our application has an interface which interpret strings such as `Command:Next`.

However, some component don't want to change their implementation only to fit the specifications of an application. For example, the HandVu system returns String like «Lpalm» which mean nothing for our ImageViewer application. Actually, we decided that «Lpalm» would mean `Command:Next`.

To achieve this mapping of Strings, we developed a new component we called Mapping. The way it works is very easy once the HandVu component is plugged to the mapping component, the mapping component read a file ( coma-dot separated) wich gives to the component the mapping of the String. For example *Lpalm;Command:Next*. For sure we could develop another mapping component which could map for example integer to String or whatever. But since our component (HandVu and Quill) didn't return any other types than String there was no need for doing those components. You can see a graphical illustration of this component on figure 61.

The goal of this mapping component is then to provide independence between messages that a component can send and message that another component can receive.

Before explaining how we integrated our component to OpenInterface, we will briefly define this mapping component :

**MappingComponent :***Figure 61: Mapping Component*

We are describing here the CIDL file of the MappingComponent. CIDL explanations were given in the section 2.8 or in [LAWS 06].

- The component own one factory where we can pass the translation file path:

```

<Factory>
  <Interface type="function">
    <Name value="MappingComponent"/>
    <Argument>
      <Param name="file">
        <PrimitiveType name="string"/>
      </Param>
    </Argument>
  </Interface>
</Factory>
  
```

*Code 3: CIDL factory code for the Mapping Component factory*

- And 2 pins : One to received input String (the sink) :

```

<Sink id="InputString_Manager">
  <Interface type="function">
    <Name value="inputString"/>
    <Argument>
      <Param name="inputString">
        <PrimitiveType name="string"/>
      </Param>
    </Argument>
  </Interface>
</Sink>
  
```



```

</Interface>
</Sink>

```

**Code 4: CIDL Sink code for the Mapping Component.**

- And another to realize the callback (the source):

```

<Source id="OutputString_Manager">
  <Callback>
    <Interface type="function">
      <Name value="newEvent"/>
      <Argument>
        <Param name="outputString">
          <Descr>Output String</Descr>
          <PrimitiveType name="string"/>
        </Param>
      </Argument>
    </Interface>
    <Setter>
      <Interface type="function">
        <Name value="setMappingEventListener"/>
        <Argument>
          <Param name="cbck">
            <Descr>Listener interface</Descr>
            <CustomType type="javaclass"
name="mappingManager.MappingEventListener"/>
          </Param>
        </Argument>
      </Interface>
    </Setter>
  </Callback>
</Source>

```

**Code 5: CIDL Source code for the Mapping Component.**

- Translation file have to look like this :

**<InputString>;<OutputString>**

And the component send **OutputString** corresponding to **InputString**

```

open;Command:Select
Lpalm;Command:Next
Lback;Command:Previous
closed;Command:CloseHandVu
victory;Command:MiniMaxi
sidepoint;Command:Reduce

```

**Code 6: HandVuMapping.txt : Translation file for HandVu**

## 5.1 OpenInterface and hand-based recognition

As for InterpiXML we use the HandVu features to write its results on a local port. So the new components need only to read messages from HandVu software on a specific port and then any else component can come to subscribe to a callback (as explain in section 2.8) to receive this gesture event under string type.

The advantages of this method (to work by local port between the recognition software and OpenInterface) is that the modality can begin before or after the OpenInterface components, can stop during execution and restart later without problem. So the modality is independent from OpenInterface.

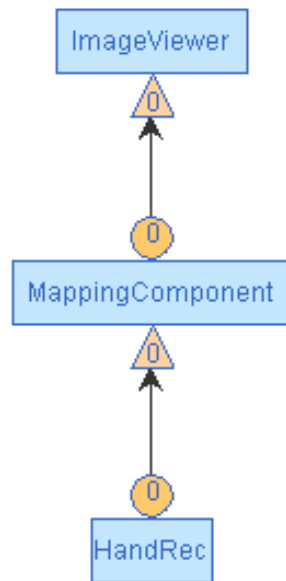
The disadvantages is that modality doesn't start automatically with the OpenInterface component.

### 5.1.1 Architecture

In fact it's not really good to talk about architecture in the case of OpenInterface. In fact we have created a specific architecture but we defined the CIDL and a PDCL for the interconnexions of the different components.

The «architecture» is then composed by 3 components :

- The final interface (ImageViewer) which have one sink to received String messages which represent a command as input.
- The HandRec component which read messages from HandVu and have callback to subscribe to received these gestures as String.
- The Mapping component which translate the words of the HandRec component to be understandable to the ImageViewer component. This translation is contained into a file.



*Figure 62: HandVu component pipe in OpenInterface*

The MappingComponent do a callback to HandRec to received gestures when they appears. And the GUI do the same to the MappingComponent to receive the translated gesture and then react to it.

### 5.1.2 Implementation

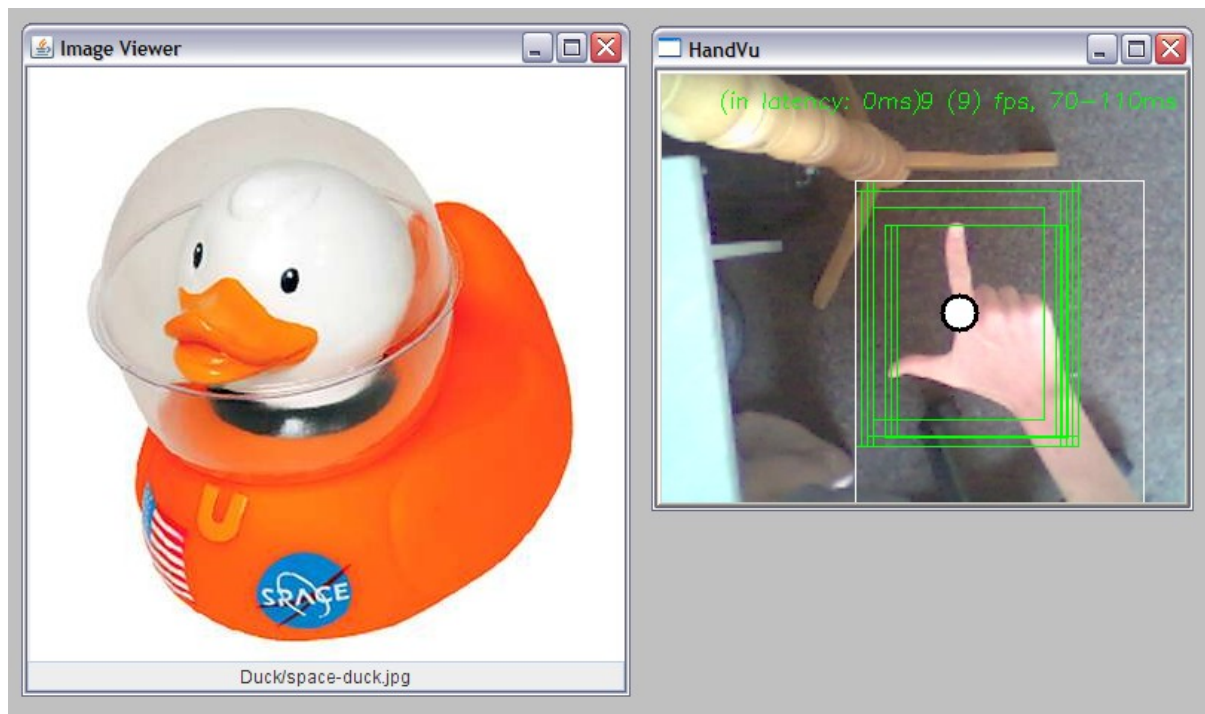
The implementation consist only in the HandRec component which is in charge to manage connection to the HandVu software, and generate event with the gesture when one is recognized. The GUI component is not specific to our work and will not explain. It just react correctly to String passed to it.

After we need only to write the CIDL files to describe each components and a PDCL file to build the pipe between each of them. All this files are presented in the appendix. Languages description can be found in [LAWS 06].

### 5.1.3 Examples

Examples in this chapter are based on an interface which is an image viewer. This image viewer has been developed during the eNTERFACE workshop and will be available on the OpenInterface website. [NetLink13]

Possibilities are to see next or previous image (duck), to choose to display file path or command on the bottom label (here the path file) and as usually to minimize-maximize, to iconify or to close the frame. On this example the HandVu software recognize the previous gesture and the image viewer will display the previous duck images.



*Figure 63: HandVu and a GUI with OpenInterface print screen*

### 5.1.4 Evaluation

The evaluation of the hand based gestures recognition in OpenInterface looks like the evaluation of the same modality in InterpiXML because for user except the way to start the modality and the interface, system behaviour is just similar. We will test in next chapter if people seems difference between these two platforms.

## 5.2 OpenInterface and pen-based recognition

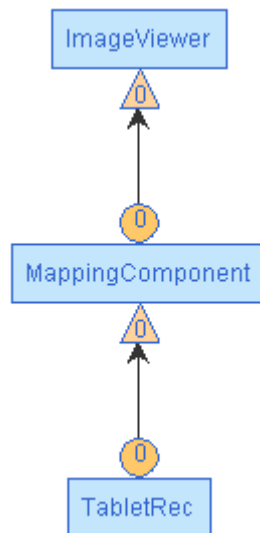
Conversely to what we did in InterpiXML, we didn't use any socket to communicate between the QuillRecognizer and the final application. Actually we are using all the power of the OpenInterface. Once a gesture is recognized into the TableRec component, a String representing the gesture is sent directly to the connected component, due to a Callback mechanism, here the mapping Component.

Also conversely to what we did in the hand camera recognizer for OpenInterface, the HandRec component is starting when we start the OpenInterface. But the corresponding disadvantage is that if we close this modality (it not means that we don't use it), all the platform closes.

### 5.2.1 Architecture

The principle of architecture is almost the same as hand-based recognition. There here neither to talk about architecture but we do prefer talking about «pipelines».

As we can see on figure 64 the way information is exchanged is quite simple. This pipeline is composed by 3 components.



*Figure 64: Quill-ImageViewer pipeline*

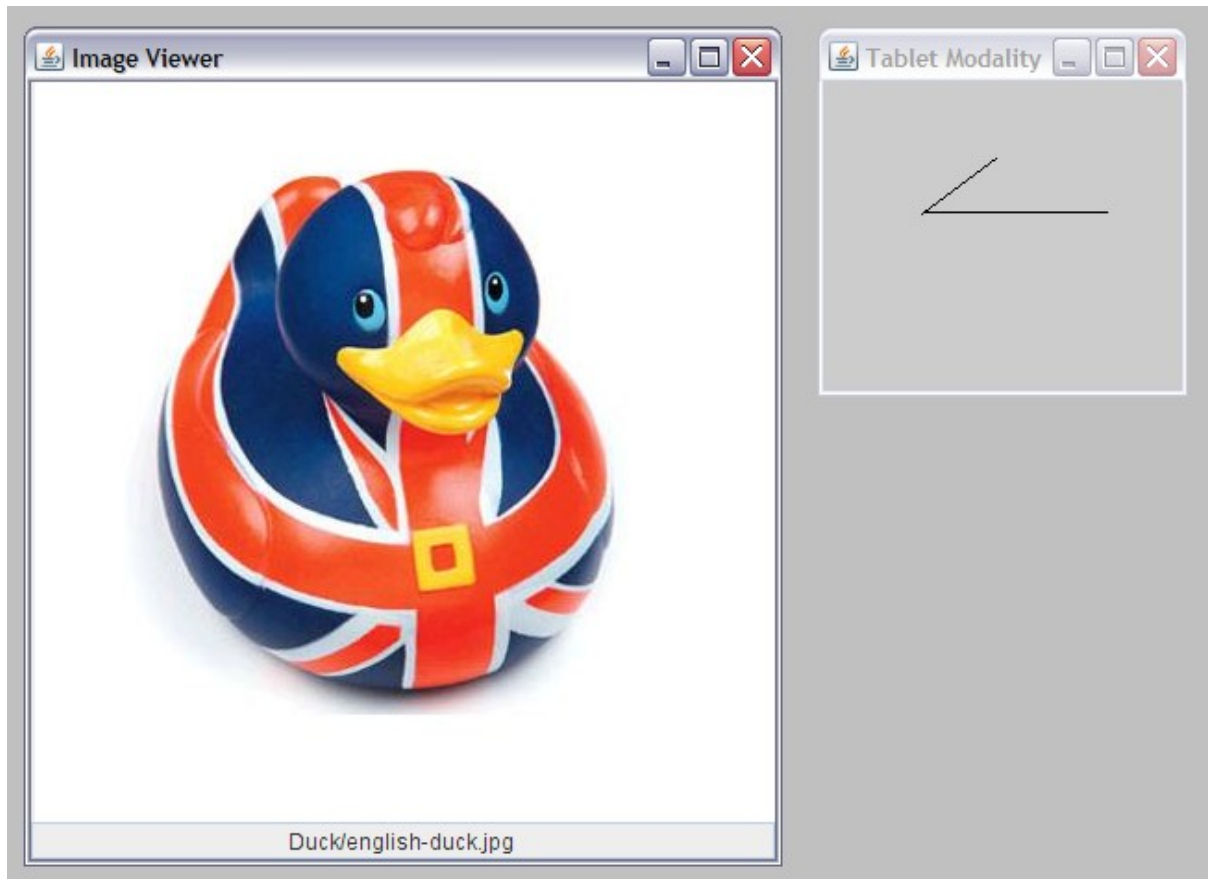
The first one is tabletRec. This component is where the recognition of the pen gesture are recognized. Once the recognition of the gesture is done (this is very fast) the mappingComponent receives the String corresponding to the gesture due to a Callback mechanism. When the translation of the mapping component as done the translation of the input String, mapping Component send directly to the application the corresponding String to the final application. Finally, the application receive a String which corresponds to a command that it can interpret.

### 5.2.2 Implementation

The implementation consisted in writing the CIDL and PDCL files. In fact, we already had our Recognizers we just had to write a HandRec Class which is sending events when pen-based gestures are recognized. As we explained in Chapter 2.8 the CIDL and PDCL we will only put the code of our CIDL and pipeline in the appendix since there's nothing really specific.

### 5.2.3 Examples

Here with the same image viewer interface, the previous pen gesture will display previous duck images on the interface.



*Figure 65: Pen based Recognition on OpenInterface*

### 5.2.4 Evaluation

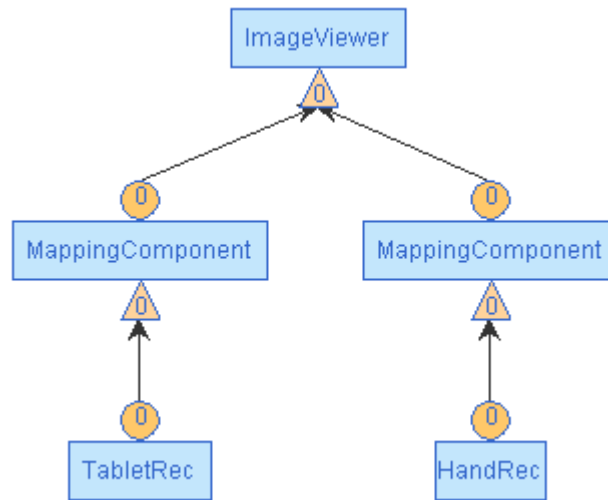
As say previous for the hand gestures section. System behaviour is just same as for InterpiXML. The only advantages of the OpenInterface platform is that when you construct the pipe (« **buildPipe pipe\_name** »), it's OpenInterface which start the application and their modalities. For this example just this line : **buildPipe pipe\_Tablet\_Mapping\_ImageViewer.xml** will launch the ImageViewer and the Tablet Modality.

## 5.3 OpenInterface with hand and pen-based recognition

We want here to develop a real multi-modal application with ImageViewer. And add the two modalities together to work with the application.

And that's very easy because it just need to modify a few the two previous integrations. The hand and the pen based modality are here totally independent.

### 5.3.1 Architecture



*Figure 66: Pipeline pen-based and hand recognition on OpenInterface*

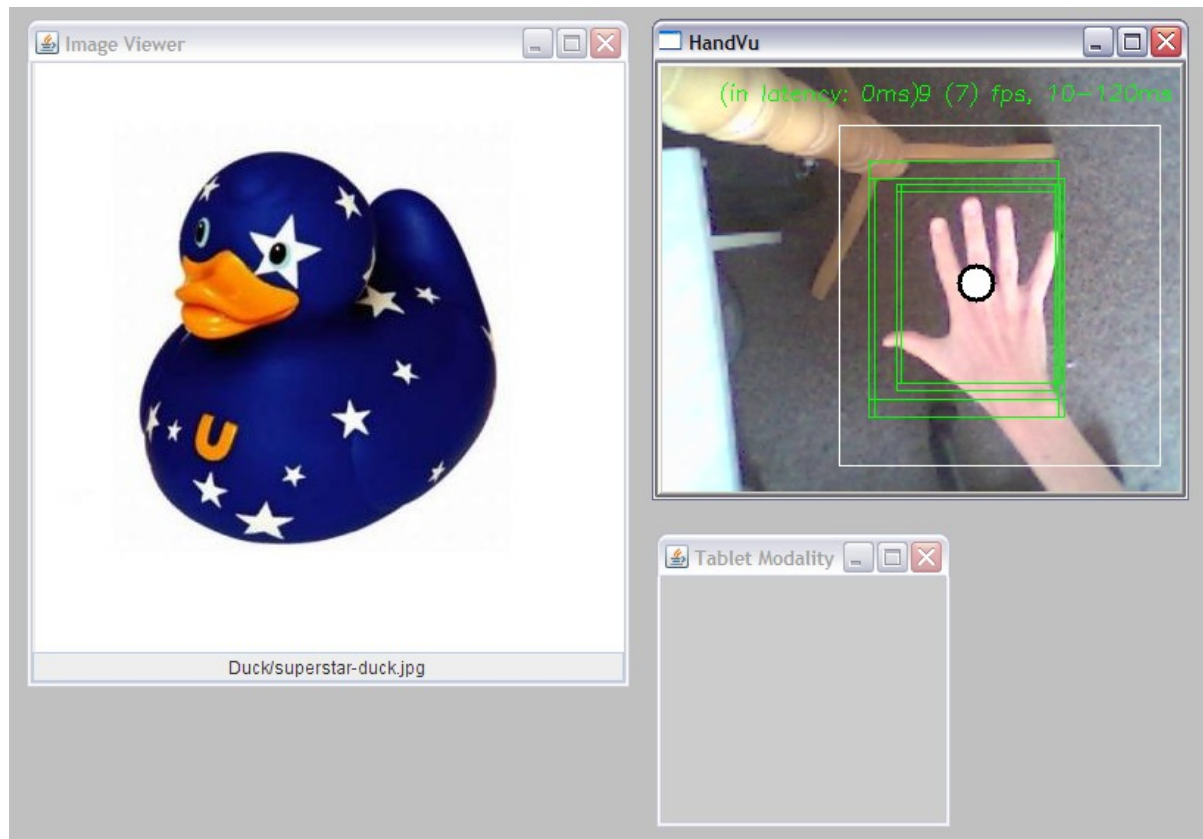
As previous we use here a MappingComponent for each modality to translate gestures from the recognition component to comprehensible gestures for the ImageViewer. These two MappingComponent give their result to the same sink on ImageViewer.

### 5.3.2 Implementation

It need here just to create a new pipe from a PDCL file which is just compose of the two previous pipe for the hand and the pen pipe.

### 5.3.3 Example

We present here the image viewer interface with the two modalities. Only HandVu software recognize a gesture which have as effect to change the label text on the interface bottom



*Figure 67: Pen-based and hand recognition on OpenInterface*

### 5.3.4 Evaluation

Here again, nothing really new appear. As said before to work with the two modalities is not easy except maybe for left-handed. We will see that in next chapter.

## 5.4 General evaluation

As for the general evaluation of InterpiXML on section 4.4 we can evaluate the CARE properties. CARE properties are fulfilled as InterpiXML that is Assignment and Equivalence since it's the same application, redundancy and complementarity doesn't take sense in this application. However, OpenInterface can produce complementarity.

When programmers describe the pipeline components, they only have to specify after the Mark-up <filter> the order of the parameters are specified.

This can be achieved by describing a more complex pipeline. In fact, the users have to specify for each function requiring parameters a <pin> mark-up. And then connecting the pins and filtering the orders of parameters received. This have already been accomplished for example the ImageNavigation\_gesture\_voice which can be found on the OpenInterface web site.



Concerning Redundancy, it's difficult since all inputs have to send the information at the same time and should provide a degree of confidence of their signals. There's here a trade-off between the good interpretation and the rapidity of the response. Because if system is waiting for 2 seconds in order to have another input signal meaning the same as the first signal despite the fact that user try to execute another command. Redundancy should concern some specific application requiring all the inputs at the same time.

## 5.5 InterpiXML integration to OpenInterface

Here come an extra section not planed at the beginning of our work. During the eNTERFACE workshop in Istanbul, we discussed a lot with the OpenInterface platform developer about their platform but also explained how the InterpiXML platform works. And during last days we had the idea to integrate InterpiXML as a simple OpenInterface component as any other modality or interface. We implemented this solutions during last Friday of the workshop and when we came back in Belgium that worked. We will explain here how does it work, first describe pipeline for the architecture of these components. Then we explain the very little change that InterpiXML has been subject in the implementation section and finally give an example with a print screen to show now to final version of InterpiXML for this thesis.

### 5.5.1 Architecture

InterpiXML is just considered here as any other interfaces which provide a sink to receive data from other modalities. The hand and pen modalities are each plug to a MappingComponent to translate string from modality to InterpiXML. And each of these MappingComponent are plug to InterpiXML to send translation command when gesture are recognized by one modality. Here are the pipe architecture.

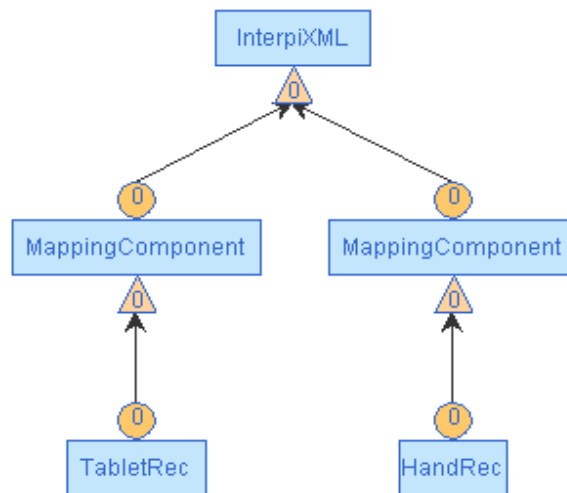


Figure 68: Pipe for InterpiXML and both modalities

### 5.5.2 Implementation

We need only to add two things to InterpiXML to implement a sink reachable from other OpenInterface components. First a factory, to build InterpiXML from the pipe :

**public static Main init()**

This method return a Main class which is class were InterpiXML is started. If InterpiXML is started from this method it will not start two threads to listen on both port 7045 and 7046. It will only listen event from the OpenInterface platform which can communicate from the method :

**public void setCommand(String cmd)**

OpenInterface modality can send String to this function. When the function is called with a string which is the command to send to the interface, a new Object named **OIEvent** is posted on the eventBus. All interfaces created have subscribed to receive this kind of event. So the eventBus will send this event to all interfaces and this one which have the focus will react to the command encapsulated in the **OIEvent** Object that it received. Command have to have same features than explain previously for the InterpiXML generated interfaces (in section 4.1.2).

### 5.5.3 Example

The example is started by command line **buildPipe pipe\_InterpiXML.xml**. InterpiXML inform user on the bottom label that OpenInterface is connected. Here both modalities can communicate with all interfaces generated by InterpiXML. All works as previously for standard InterpiXML release except that here the pipe run automatically the tablet modality. And if you close the tablet modality, InterpiXML stop also and if you close InterpiXML, tablet modality stop.

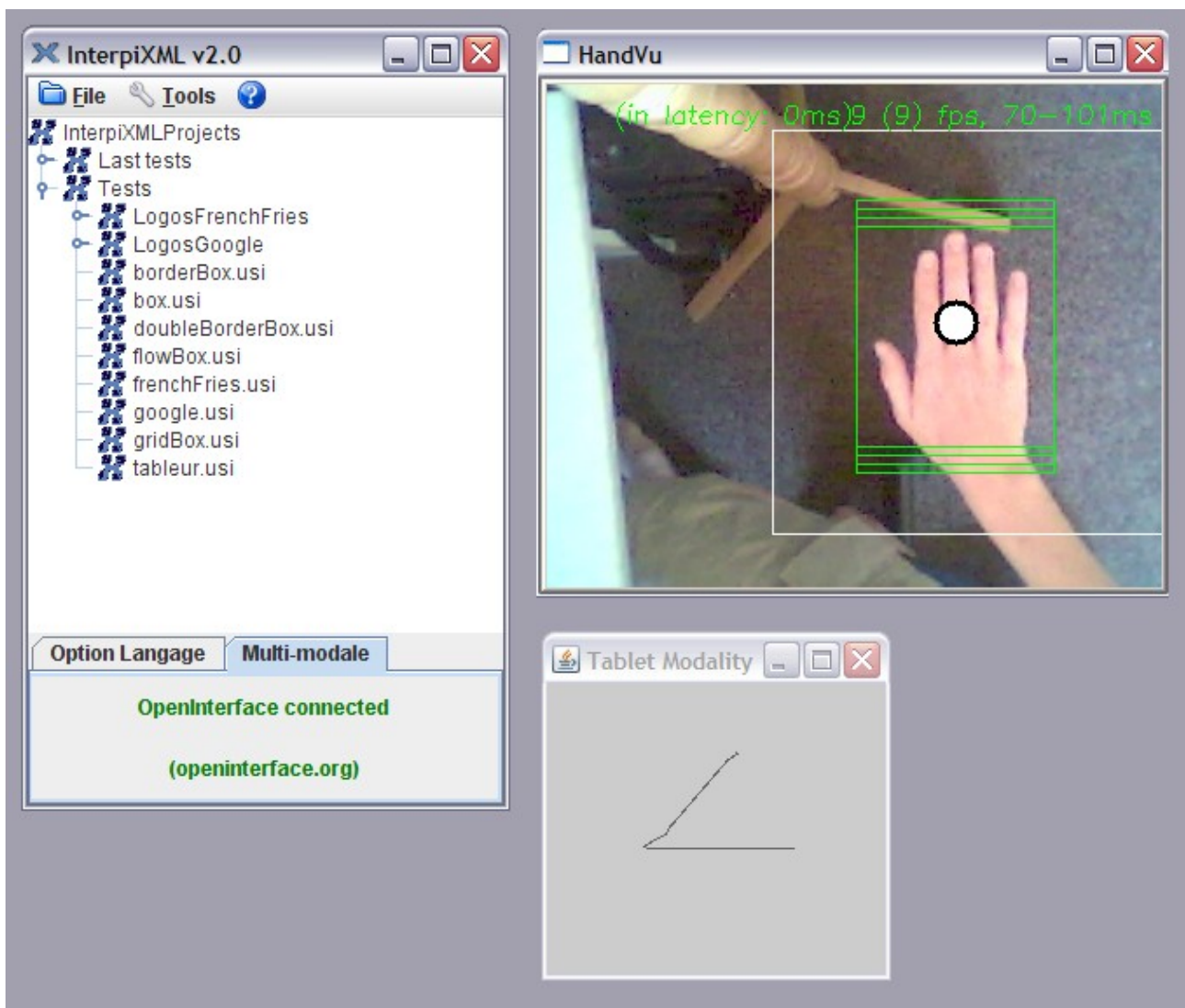


Figure 69: InterpiXML - OpenInterface connection with 2 modalities

## 5.5.4 Evaluation

This version presents a great advantage for developers because now you can add a modality only by modifying the pipe (**pipe\_InterpiXML.xml**). It's for example very easy to add the speech recognizer with a MappingComponent between it and InterpiXML.

No test has been realised on this « multi-platform » but we think that users will not see differences between this and previous releases where modality was added in the InterpiXML code itself.

## 6. Tests

After developing our modalities and tested them on experimented people (us) we did an experiment to confort what we said. To have a real and objective evaluation, we did a short experiment on few volunteers.

### **The goals :**

The goals of the experiment was to evaluate the utilisability. We evaluated the usability of our modalities that's pen-based gesture and hand camera gesture and then tested if there was a impact depending on platform that is InterpiXML and OpenInterface.

To evaluate utilisability, we will measure some criterion like time to execute a task, the fulfilment of the task and user evaluation after using the two devices. Those evaluations are done for both platforms. To evaluate our modalities we will draw Likert squale on different criterion.

### **The experiment itself :**

The experiment consist for users to fill a simple form by using pen-based gesture and hand-based gestures. The task is to insert «joel» for name, to select "Mayonnaise", to select "Moyenne" and finish with a click on "Ok" button. Here you can see on figure 70 the task model of the interface produced with Ideal2W tool. And then on figure 71 the interface itself.

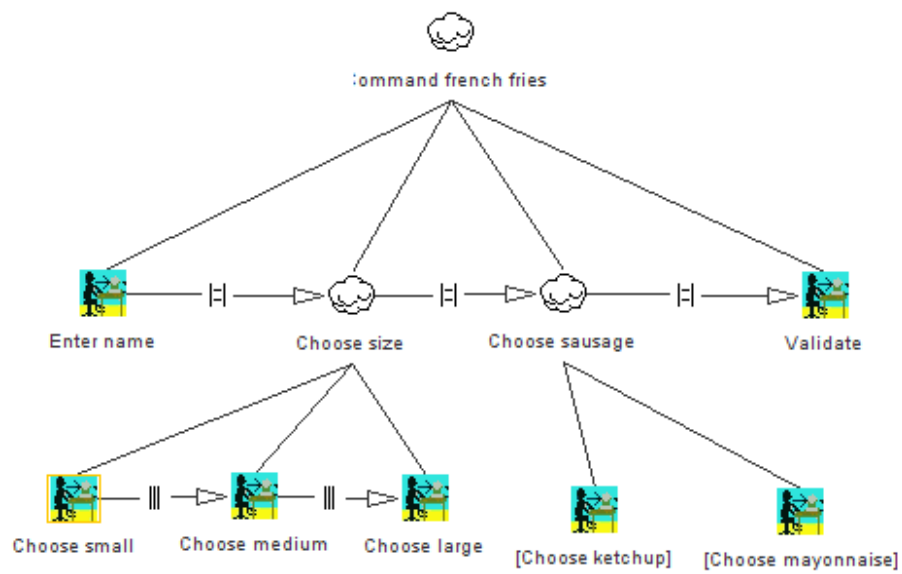


Figure 70: Task model for test interface

The screenshot shows a window titled "Commande de frites" (French fries order form). The main heading is "Commandez vos frites" (Order your fries). Below the heading is a photograph of two golden-brown french fries. Under the photo is a text input field labeled "Nom" (Name). Below the name field are two sections for sauce and size. The "Sauce(s) :" section has two checkboxes: "Mayonnaise" and "Ketchup". The "Taille :" (Size) section has three radio buttons: "Petite", "Moyenne", and "Grande". At the bottom of the form are two buttons: "Ok" and "Annuler" (Cancel).

Figure 71: French fries order form

**The experiment protocol :**

The experiment orders at follow. First volunteers are welcomed and they fill a demographic form (shown on appendix) and then receive a primary informations about the topic of the experiment. They also receive some informations about gestures. We insisted on the fact that if the system doesn't work it's system's fault not their fault. They are informed how will the experiment be carried on.

After, this short explanation, volunteers see the devices and try gesture for 10 minutes on another interface see figure for the learning interface. In fact 5 minutes for each devices. The gestures are shown on the wall where they do the test see installation on figure 73 The test is recorded with a digital camera.



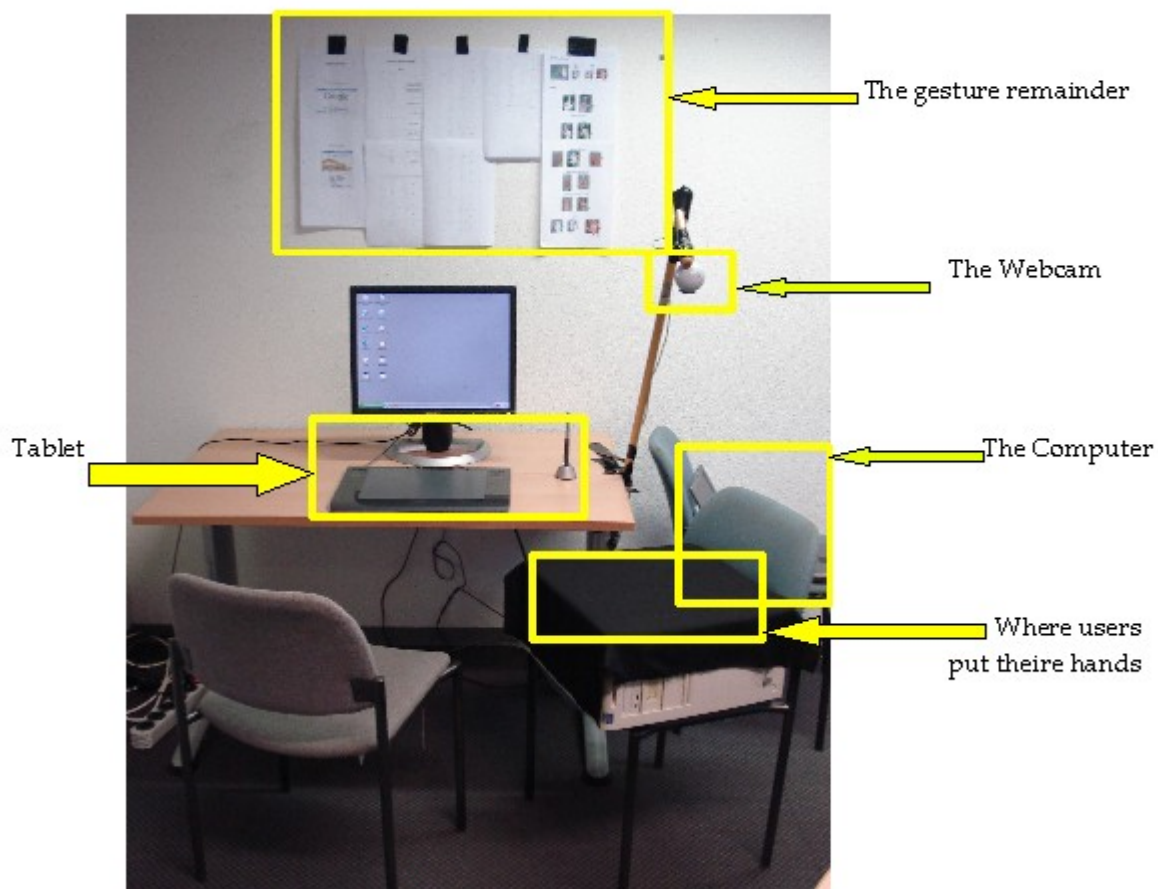
*Figure 72: The learning Interface*

In fact we can divide the test in two parts. First, the test is done with InterpiXML platform. Users have to fill the form presented on figure 71 only with the hand-based gesture (without having to fill the field name) when it's done, we restart a new form that they only have to fill with pen-based gesture (here they must fill the name with «joel») and finally when finished, they have to fill once again the form with the modality they want to use including the use of both modalities or not (here again they have to fill the name with «joel» value).

After those tests on InterpiXML the set of form filling is re-done but using OpenInterface. Some stuffs has to be noticed is that if the users takes more than 5 minutes to fill a form or do any wrong operation that make the system exit, then they are stopped and have to carry on to the following test. When users have completely finished the test, they fill a form for evaluating the modalities and their degree of satisfaction. Then, they are thanked and receive a chocolate.

### The experiment conditions :

On figure 73, we can have a look at the room where experiment held. As we can see the user sit in front of a 19" screen on which interfaces are projected. The laptop on the right is commanded by an supervisor to launch various interfaces. We have also set a «device» for helping users not to lost themselves in space for the hand position. Pictograms representing gestures are also put in front of users because as the purpose of this experiment is not to define whether gestures are well-designed or not, we don't evaluate the learning of gestures themselves. The temperature room was a bit too warm.



*Figure 73: The experiment room*

### The evaluation forms:

The forms users had to fill before and after the test are given in annexes. After the tests we first evaluate modalities concerning 4 criterion. The general appreciation, the ability to move into the interface, the ability to select an item and finally only for pen-based gesture, the ability to insert text. Those questions are evaluated from 1 to 7. 7 being the best result. Then the form asks questions about general questions in parallel for both systems like the ability to finish the tasks, the learning evaluation etc. (see appendix for complete questions).

The forms we distributed to the volunteers are forms which has been developed by IBM. Those form are quiet reliable to evaluate the usability of an interface since those after scenarios questionnaire has excellent internal consistency, with coefficient alphas across a set of scenarios ranging from 0,9 to 0,96. [NetLink14]

### **The pre-test:**

We achieved a pre-test on a volunteer to affine our protocol. In fact we saw that we had to show to volunteers how to use hand-recognition and show an example what we haven't done for the first volunteer. We also defined a maximum time of 5 minutes per interfaces instead of the 10 estimated. Volunteers are getting bored if after 5 minutes they don't achieve their tasks.

### **The participants :**

We gathered a sum of 14 participants. We collected 11 man and 3 women. Average age is 24,4 but extremes are 19 and 37. Most part of them are used to computer and evaluate there ability to manage the mousse on 6,14 on a level from 1 to 7. Concerning the tablet and the camera ability, respectively 2,36 and 1,54 which means that those users are not used to those devices.

### **Hypothesis :**

Before achieving the evaluation campaign on our volunteers, we thought about some hypothesis :

1. Volunteers should not see the differences between OpenInterface test interface and InterpiXML interface. As those interfaces are exactly similar and users won't be able to feel any differences (those differences are only at an implementation sight).
2. Tablet should be preferred to webcam because stylus is more like mouse and users have the habit to use this device.
3. Webcam accuracy may disappoint users. In fact HandVu library use to take some time to recognize the hand. Once this hand is recognized, it's quite fast to recognize gesture.
4. Due to our modalities no users will use simultaneously both modalities.
5. Experimented users of tablet should accomplish task more rapidly.
6. Learning is very important for the both modalities.

### **The Results and conclusions :**

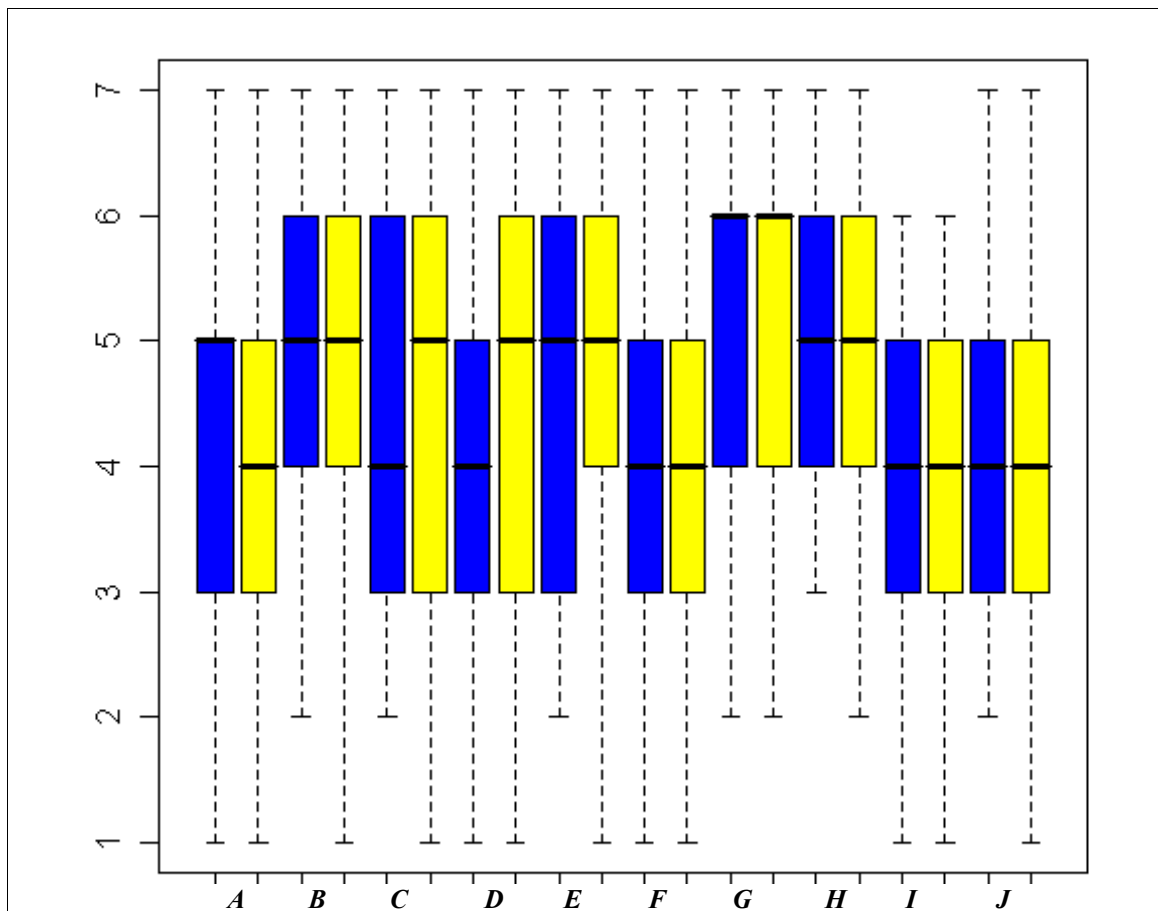
We will compare the results we got in comparison with our hypothesis.

#### ***1) Volunteers don't see the differences between OpenInterface and InterpiXML***

To evaluate if OpenInterface platform and InterpiXML were perceived the same by user, we realised with R (powerful statistical tool) [NetLink15] a chart-box comparing the answers of the volunteers for InterpiXML and OpenInterface. We took into account the generals questions (those in the second part of the form, see appendix).



The result of this box-charts are shown on figure 74. What can be interpreted from this chart, is that as we expected users don't seems to see any differences between the two platforms. We can see in blue the answers to questions for InterpiXML and in yellow to OpenInterface. We see that 50 % of the users answered the same results for OpenInterface and InterpiXML excepted for questions D and E where users slightly prefers OpenInterface. Maybe we can interpret those results as a learning effect from the users since OpenInterface was done at the second phase of the test. Seeing those results, we conclude that there is no significant differences between the 2 platforms.

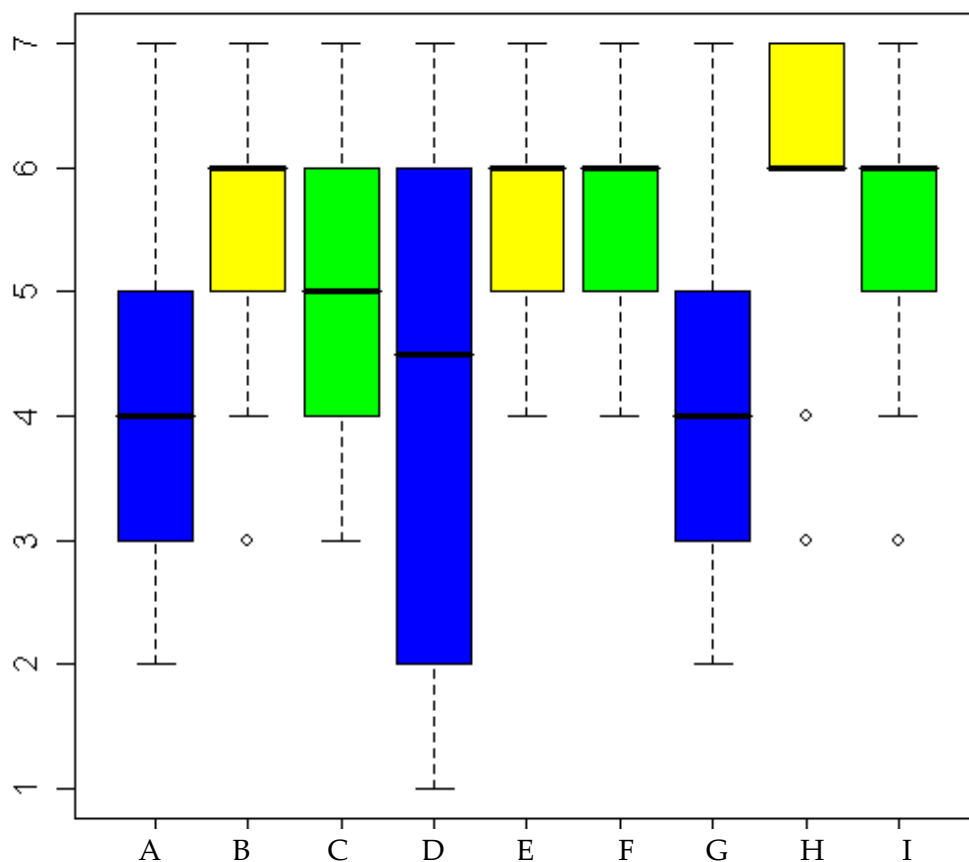


*Figure 74: InterpiXML and OpenInterface Users comparaison*

- A: En général, je suis satisfait(e) de la facilité d'utilisation de ce système.  
 B: Ce système est simple à utiliser.  
 C: J'ai complété mon travail correctement en utilisant ce système.  
 D: J'ai été en mesure de compléter rapidement ma tâche avec ce système.  
 E: J'ai complété mon travail efficacement en utilisant ce système.  
 F: Je me sens à l'aise avec ce système.  
 G: J'ai eu de la facilité à apprendre comment utiliser ce système.  
 H: Je crois être devenu(e) rapidement efficace en utilisant ce système.  
 I: Ce système possède toutes les fonctions et le potentiel correspondant à mes attentes.  
 J: En général, je suis satisfait(e) de ce système.

## 2) Volunteers should prefer tablet to webcam

Tablet is a more common device since it emulates a bit the mouse device. Take a look at the figure 75, yellow boxes represents the tablet, blues boxes the webcam and finally greens boxes the both modality together. We see that tablet is effectively preferred to the camera since almost all the yellows boxes are above the blues ones. But what is really interesting here is that when users have the choice of the modality, the global appreciation joins the evaluation of the tablet. This means that tablet is playing the role of a moderator. Tablet is moderating the depreciation of webcam. As we said in the introduction, the weakness of a modality are overcomes by another modality. This can be explained by the fact that when volunteers have the choice of modality, 46 % of them choose to only use the tablet and 50 % of them use both and 4% uses only camera (see on figure 76). So the preference increase as they are using tablet or combining tablet with camera.



*Figure 75: Appreciation*

A : Comment avez-vous apprécié ? Webcam

B : Comment avez-vous apprécié ? Tablette

C : Comment avez-vous apprécié ? Les 2 modalités ensemble

D : Comment évaluez-vous la manière de se déplacer dans l'interface ? Webcam

E : Comment évaluez-vous la manière de se déplacer dans l'interface ? Tablette

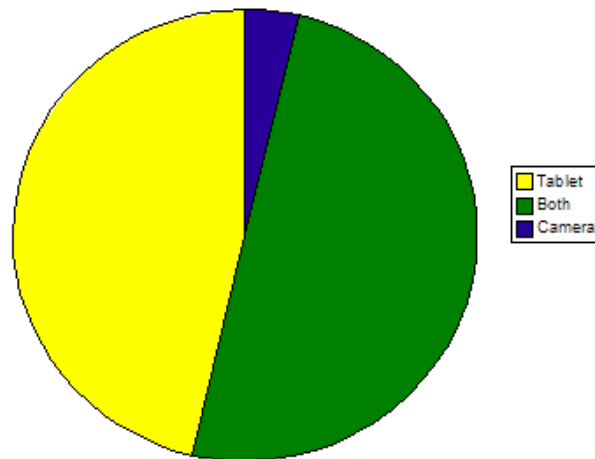
F : Comment évaluez-vous la manière de se déplacer dans l'interface ? Les 2 modalités

G : Comment évaluez-vous la manière de sélectionner un élément ? Webcam

H : Comment évaluez-vous la manière de sélectionner un élément ? Tablette

I : Comment évaluez-vous la manière de sélectionner un élément ? Les 2 modalités

### Modality Usage when choice offered



*Figure 76: Modality used when choice offered*

#### **3) Webcam accuracy may disappoint users**

The only proof of this hypothesis is the comments users made at the end of the form. 50 % of the volunteers were disappointed not be the webcam itself the but by the time the webcam take to focus on the hand.

#### **4) Volunteers don't use the 2 modalities simultaneously**

As we shown in the point 2, 50 % of users are using both modality but not exactly at the same time. They have to put the stylus done or take off the hand of the camera to take the stylus in order to use the other device. Although, we had the very interesting case of a user who was left handler and could manage simultaneously both modalities (see video "*test06.wmv*" on joint CD). He navigates on the interface with the camera and selected items with the tablet. But it represent only one person on 14 an only  $\frac{1}{4}$  of the left-handed persons who passed the test. We should have a larger number of left-handed persons to validate or invalidate this hypothesis.

#### **5 ) Experimented users of tablet should accomplish task more rapidly**

To answer this hypothesis, we performed a correlation test using the Spearman statistics with the help of R. This tests informed us that there was no correlation between the habit of the tablet and the speed that the task have been performed for the first and the second time. Results are :

---

## 6. Tests

---

Use of tablet and time to accomplish First task

```
> cor.test(corr[,1],corr[,3])
```

Pearson's product-moment correlation

data: corr[, 1] and corr[, 3]

t = -0.231, df = 11, p-value = 0.8216

alternative hypothesis: true correlation is not equal to 0

95 percent confidence interval:

-0.5975848 0.5006762

sample estimates:

cor

**-0.06947484**

Use of tablet and time to accomplish 2<sup>nd</sup> task

```
> cor.test(corr[,2],corr[,3])
```

Pearson's product-moment correlation

data: corr[, 2] and corr[, 3]

t = -0.1444, df = 11, p-value = 0.8878

alternative hypothesis: true correlation is not equal to 0

95 percent confidence interval:

-0.5805775 0.5199393

sample estimates:

cor

**-0.04351092**

This hypothesis is not quiet reliable since the number of persons estimating they use frequently a tablet is poor. However those results would lead in the fact that any users even not using often a graphical tablet could accomplish the task. Moreover, our hypothesis was false, expert tablet users are not advantaged to new users.

### 6 ) *Learning is quiet important for both modalities*

We taught that it may have some learning effect for the two modalities. In fact we drawn the time for achieving the tasks see in figure . If we compare the time required to accomplish the task, we see that the time required for InterpiXML is much higher than the one for OpenInterface. Actually when users are doing the test on the OpenInterface platform, they are experienced so they achieve faster there tasks. Time reducing goes from 30 % for the camera to 50 % for the tablet. As we can see on figure 77. Task achieving have also an impact after some learning (see figure 78). We can effectively say that learning effect is well present for both modalities.

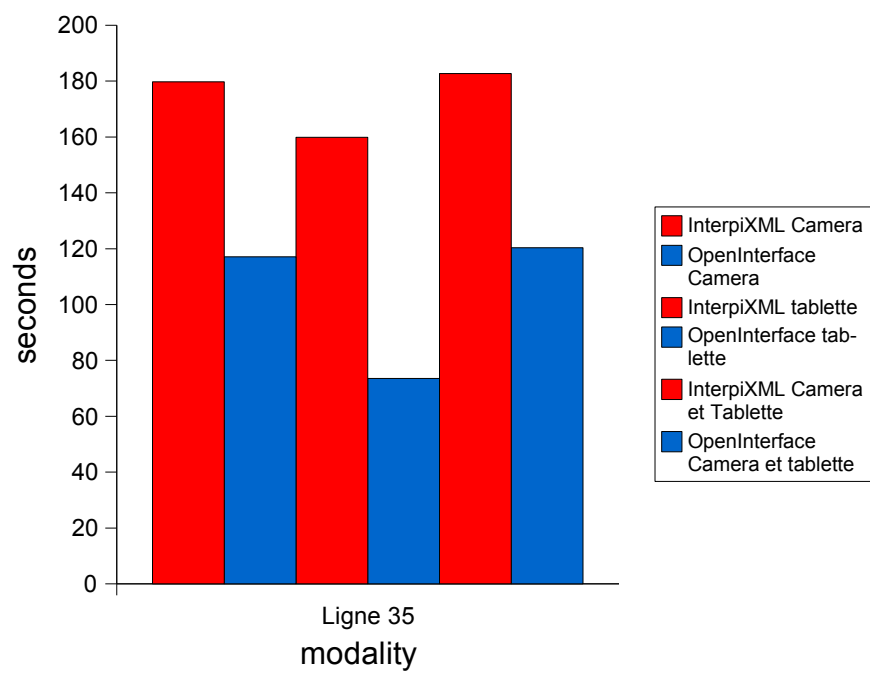


Figure 77: Time to accomplish the task

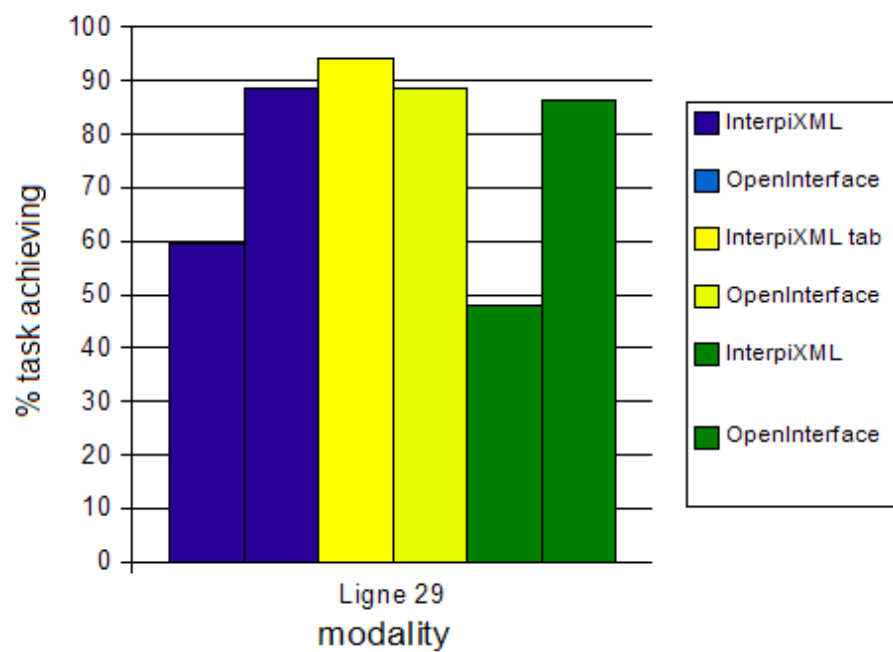


Figure 78: Task achieving

### *Other conclusions and interesting results*

We found some other interesting conclusions by achieving this experiment. First of all users are in average more than 85 % able to finish their task in between 80 and 120 seconds. This is quiet encouraging because it works and users seems to be pleased to use new interacting devices.

Secondly, there's no need for users to know how to use device to perform the task and the learning effect is present since the time to accomplish the task is decreasing fast and accomplishment rate high.

Thirdly, tablet if preferred to webcam and is more fast but we think that is only because tablet is a more natural device (emulate mouse) and because response time of webcam is low because time is needed to recognize gestures.

Fourthly, another conclusion is that it's almost impossible to use both modalities at the same time except for left-handed since both modalities require right hand.

Another conclusion is that during our experimentation we confirmed firmly what shanon Oviatt told in this paper [OVIA 02]. Especially what said in section where we explained the advantages of multimodality section 1.1.2. "When theory is joining practice..."

Gestures were sometimes confused both for camera and pen-based gestures. Mainly, two gestures were confused with camera recognition open and Lplam gesture. This confusion depended on the position of the auricular finger. Pen based gestures characters were confused and, depending on the users, were difficult to draw. In fact the result provided by the evaluation form told that users found at 40 % the insertion of text easy so were a bit disappointed.

Others things we have to notice is that some gestures leads to problems. Actually sometimes the validation gesture of the tablet were interpreted as a close action. Maybe this gesture should have been design with more care even if the major responsibility is to the Quill recognizer. Another gesture leaded to problems the closed hand gesture has 2 meanings. In fact, it serves to recognize the hand and close the windows. However to avoid those problems we took our precautions. The close hand gesture had to be recognized three times before executing the closing of the window and the close gesture with the tablet had to be recognized with a confidence degree more than 95%. However this not look sufficient.

## **Conclusions**

Time as come to conclude this whole thesis. As you can see, the chapter 2 enabled to inform us about the current state-of-the-art to find the best toolkits and best gestures for achieving our implementation. During the following phase we reflected about the possible architectures for multimodal platforms and designed all the different pen and hand-based gestures we need for our work. The key-part of this work was then to integrate both modalities on both platforms as described on chapter 4 and 5. Chapter 4 explain our architecture design choice to upgrade InterpiXML for multimodality. This work finish with a comparative analysis of the two platforms. If we had to sum our work we could say that we traversed the whole lifecycle of the software crossing design, implementation, test and evaluation.

The final evaluation we performed on the last chapter gave us the opportunity to evaluate the strengths and weaknesses of our different implementations. Our main weakness was that some gestures were either difficult to realise for new users nor were not in adequation with the action they were representing. More precisely, this implied problems to learn or to execute those gestures both for pen and hand gestures. Designing pen-based gestures was not that easy even with a tool like Quill and we could not add other gestures to HandVu at the moment. Despite this weakness, we hadn't at any time have to change our architecture. Furthermore, the way this architecture is done enable to add new modalities to InterpiXML easily. However the complete achievement of this thesis is introduced in section 5.5, we integrated InterpiXML as a component of OpenInterface. This way, any new modality can be used in InterpiXML by the use of OpenInterface.

The work which has been accomplished can be improved by adding new modalities to InterpiXML, to OpenInterface or the easiest : to InterpiXML plugged on OpenInterface because it only consist in describing a new pipeline. Obviously future work can also consist in developing more efficient toolkits for gesture recognition. We hope that this work will respond to the expectations of the readers and could provide any help to developer of this exciting research field that is multimodality .





## Bibliography :

### Books, periodics et papers :

- [BOLT 80] BOLT R. A., "Put-that-there": Voice and gesture at the graphics interface," in 7th annual International Conference on Computer Graphics and Interactive Techniques, Seattle, United States, 1980, pp. 262 – 270
- [BOUI 02] BOUISSET S., *Biomécanique et physiologie du mouvement*, Mason Paris, 2002.
- [CARD 80] CARD S. K., MORAN T. P., and NEWELL A., *The keystroke-level model for user performance time with interactive systems*. Communications of the ACM, 23(7):601–613, 1980.
- [CHAT 98] CHATTY S. and LECOANET P., *Pen computing for air traffic control*. In *Human Factors in Computing Systems*, (SIGCHI Proceedings), pages 87-94. ACM, Addison-Wesley, April 1996.
- [COUT 93] COUTAZ J, NIGAY L., SALBER D., BLANDFORD A., MAY J. and YOUNG R. , *Four Easy Pieces for Assessing the Usability of Multimodal Interaction : The CARE properties*, Proceedings of Interact'95, K. Nordby, P.H. Helmersen, D.J. Gilmore and S. Arenesen eds, Chapman&Hall, Norway, p 115-120.
- [FRAN 95] FRANKISH C., HULL R., and MORGAN P., *Recognition accuracy and user acceptance of pen interfaces*. In *Human Factors in Computing Systems*, SIGCHI Proceedings, pages 503-510. ACM, Addison-Wesley, April 1995.
- [GART 05] GARTNER inc., *Hype Cycle for computer interaction*, 2005
- [GART 06] GARTNER inc., *Hype Cycle for computer interaction*, 2006
- [KEND 90] KENDON A., *Conducting Interaction: Patterns of behavior in focused encounters*. Cambridge University Press, Cambridge, 1990.
- [KOLS 04] KÖLSCH M., *Vision Based Hand Gesture Interfaces for Wearable Computing and Virtual Environments*. Ph. D. Dissertation, August 2004.
- [LAND 93] LANDAY J. and MYERS B., *Extending an existing user interface toolkit to support gesture recognition*. Proceedings of INTERCHI '93 : Human Factors in Computing Systems, pages 24-29. ACM, Addison Wesley, April 1993.
- [LAWS 06] LAWSON L., *User Manual : OpenInterface CIDL and PDCL Specification*, TELE, Université catholique de Louvain, Belgium, July 2006.
- [LIMB 04a] LIMBOURG Q., VANDERDONCKT J., *Transformational Development of User Interfaces with Graph Transformations*, in Proceedings of the 5<sup>th</sup> International Conference on Computer-Aided Design of User Interfaces CADUI'2004, Madeira, January, 14-16, 2004, Kluwer Academics Publishers, Dordrecht, 2004.
- [LIMB 04b] LIMBOURG Q., *Multi-Path Development of User Interfaces*, PhD thesis, University of Louvain, November, 2004.

- [LONG 98] LONG A. C. Jr., LANDAY J. A., and ROWE L. A., *PDA and Gesture Use in Practice: Insights for Designers of Pen-Based User Interfaces*, CSD-97-976, EECS Department, CS Division, University of California, Berkeley, January 1998
- [LONG 99] LONG A. C., LANDAY J. A., ROWE L. A., *Implications for a gesture design tool*, Electrical Engineering and Computer Science Department, University of California at Berkeley, Berkeley, 1999
- [LONG 99b] LONG A. C., LANDAY J. A., ROWE L. A. and MICHIELS J., *Visual Similarity of Pen Gestures*, EECS Department University of California, Berkeley Technical Report No. UCB/CSD-99-1069, 1999.
- [LONG 01] LONG A. C., LANDAY J. A., ROWE L. A., *Those look similar!*, issues in automating gesture design advice, Proceedings of the 2001 workshop on Perceptive user interfaces PUI '01, November 2001
- [LONG 01b] LONG A. C., *Quill: A gesture design tool for pen-based user interfaces*, Ph.D. Thesis, Berkeley: University of California, 2001.
- [MACK 97] MACKENZIE I. S. and ZHANG S., *The immediate usability of Graffiti*, Proceedings of Graphics Interface '97, pp. 129-137. Toronto: Canadian Information Processing Society, 1997.
- [MCNE 82] McNEILL D. and LEVY E., *Conceptual Representations in Language Activity and Gesture*, pages 271-295. John Wiley and Sons Ltd, 1982.
- [MONE 06] MO Z., NEUMANN U., *Lexical gesture interface*, CGIT Lab, University of Southern California, 2006
- [MOYL 05] *The design and evaluation of a flick gesture for 'back' and 'forward' in web browsers* February 2003 Proceedings of the Fourth Australasian user interface conference on User interfaces 2003 - Volume 18 AUIC '03
- [NIGA 95a] NIGAY L., COUTAZ J., SALBER D., BLANDFORD A., MAY J., YOUNG R. M., *Four easy pieces for assessing the usability of multimodal interaction: the CARE properties*, INTERACT 95, 1995
- [NIGA 95b] NIGAY L., *MATIS : un système multimodal d'information sur les transports aériens*, 1995
- [OVIA 02] OVIATT S. L., *Breaking the Robustness Barrier: Recent Progress on the Design of Robust Multimodal Systems*, in Advances in Computers, vol. 56, M. Zelkowitz, Ed., Academic Press, 2002
- [OVIA 04] OVIATT S., SENEFF S., *Introduction to mobile and adaptive conversational interfaces*, ACM Transactions on Computer-Human Interaction (TOCHI), v.11 n.3, p.237-240, September 2004
- [RUBI 91] RUBINE D., *The Automatic Recognition of Gestures*. PhD thesis, School of Computer Science, Carnegie Mellon University, CMU-CS-91-202 1991. 660, 1991
- [RUGE 03] RUGELBAK and HAMNES K., *Multimodal Interaction – Will Users Tap and Speak Simultaneously?*, Teletronikk, 2003

- [STAN 07] STANCIULESCU A., *A Design Space for Developing Multi modal User Interfaces of Information Systems*, PhD thesis, University of Louvain, June, 2007
- [TIAN 06] TIAN F., CHENG T., WANG H., DAI G., *Research on User-Centered Design and Recognition Pen Gestures*, Computer Graphics International 2006: 312-323, 2006
- [UIMS 92] The UIMS Tool Developers Workshop, *A metamodel for the runtime architecture of an interactive system*, SIGCHI Bulletin, January 1992
- [WEST 03] WESTEYN T., BRASHEAR H., ATRASH A., and STARRNER T. , « Georgia Tech Gesture Toolkit: Supporting Experiments in Gesture Recognition », College of Computing, Georgia Institute of Technology

## Web links :

- [NetLink01]  
[http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL\\_COPIES/COHEN/gesture\\_overview.html](http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/COHEN/gesture_overview.html)
- [NetLink02] <http://optimoz.mozdev.org/gestures/defaultmappings.html>
- [NetLink03] <http://www.youtube.com/watch?v=kKvoFeGFOOQ>
- [NetLink04] <http://www.omniglot.com/writing/graffiti.htm>
- [NetLink05]  
[http://www.microsoft.com/france/windows/xp/tabletpc/images/doc/part2\\_autodemo.swf](http://www.microsoft.com/france/windows/xp/tabletpc/images/doc/part2_autodemo.swf)
- [NetLink06] <http://www.apple.com/fr/macosx/overview>.
- [NetLink07] <http://www.microsoft.com/downloads/details.aspx?FamilyId=B46D4B83-A821-40BC-AA85-C9EE3D6E9699&displaylang=en>
- [NetLink08] <http://Quill.sourceforge.net/>
- [NetLink09] <http://www.movesinstitute.org/~kolsch/HandVu/HandVu.html>
- [NetLink10] <http://www.similar.cc>
- [NetLink11] <http://www.enterface.net/whatare.html>
- [NetLink12] <http://www.usixml.org/index.php5?mod=pages&id=8>
- [NetLink13] <http://www.openinterface.org>
- [NetLink14] <http://www.acm.org/perlman/question.cgi?form=CSUQ>
- [NetLink15] <http://www.r-project.org>

## Appendix

*Demographic forms***Formulaire d'enquête****Données personnelles :**

Code Personnel (initiales) : .....

Sexe : ☐ Homme ☐ Femme

Age : ..... ans

**Profession :**☐ Etudiant☐ Cadre☐ Sans emploi☐ Indépendant☐ Autre : .....☐ Ouvrier☐ Profession libérale☐ Employé☐ Retraité

Domaine d'activité : .....

**Enquête :**J'utilise un ordinateur : **Jamais** 1 2 3 4 5 6 7 **Tous les jours**

J'utilise un ordinateur avec le système :

☐ Windows☐ Linux☐ Mac OS☐ Autre .....J'utilise la souris d'une manière : **Imparfaite** 1 2 3 4 5 6 7 **Parfaite**J'utilise une *webcam* : **Jamais** 1 2 3 4 5 6 7 **Souvent**J'utilise une *tablette graphique* : **Jamais** 1 2 3 4 5 6 7 **Souvent**

J'utilise d'autres périphériques :

..... **Jamais** 1 2 3 4 5 6 7 **Souvent**..... **Jamais** 1 2 3 4 5 6 7 **Souvent**..... **Jamais** 1 2 3 4 5 6 7 **Souvent**

*Les informations recueillies lors de cette expérimentation seront anonymisées, exclusivement utilisées dans le cadre de cette étude et ne seront, en aucun cas, divulguées à d'autres fins.*

**Signature :**

*Evaluation forms***Debriefing**

Code Personnel (initiales) : .....

Sélectionnez un nombre entre 1 (très bon) à 7 (très mauvais) pour répondre à chaque question.

**Comment avez-vous apprécié ?**

		Très mauvais							Très bon
InterpiXML	Webcam :	1	2	3	4	5	6	7	
	Tablette graphique :	1	2	3	4	5	6	7	
	Les deux :	1	2	3	4	5	6	7	
Open Interface	Webcam :	1	2	3	4	5	6	7	
	Tablette graphique :	1	2	3	4	5	6	7	
	Les deux :	1	2	3	4	5	6	7	

		Très mauvais							Très bon
InterpiXML	Webcam :	1	2	3	4	5	6	7	
	Tablette graphique :	1	2	3	4	5	6	7	
	Les deux :	1	2	3	4	5	6	7	
Open Interface	Webcam :	1	2	3	4	5	6	7	
	Tablette graphique :	1	2	3	4	5	6	7	
	Les deux :	1	2	3	4	5	6	7	

**Comment évaluez-vous la manière de sélectionner un élément ?**

		Très mauvais							Très bon
InterpiXML	Webcam :	1	2	3	4	5	6	7	
	Tablette graphique :	1	2	3	4	5	6	7	
	Les deux :	1	2	3	4	5	6	7	

Open Interface	Webcam :	1	2	3	4	5	6	7
	Tablette graphique :	1	2	3	4	5	6	7
	Les deux :	1	2	3	4	5	6	7

Comment évaluez-vous la manière d'insérer du texte ?

		Très mauvais				Très bon			
InterpiXML	Tablette graphique :	1	2	3	4	5	6	7	
Open Interface	Tablette graphique :	1	2	3	4	5	6	7	

### Questions sur l'évaluation globale :

Selectionnez un nombre suivant une échelle de réponse allant de 1 signifiant le désaccord total à 7 signifiant l'accord parfait :

Désaccord    1   2   3   4   5   6   7    Accord

		InterpiXML							Open Interface						
1.	En général, je suis satisfait(e) de la facilité d'utilisation de ce système.	1	2	3	4	5	6	7	1	2	3	4	5	6	7
2.	Ce système est simple à utiliser	1	2	3	4	5	6	7	1	2	3	4	5	6	7
3.	J'ai complété mon travail correctement en utilisant ce système.	1	2	3	4	5	6	7	1	2	3	4	5	6	7
4.	J'ai été en mesure de compléter rapidement ma tâche avec ce système	1	2	3	4	5	6	7	1	2	3	4	5	6	7
5.	J'ai complété mon travail efficacement en utilisant ce système	1	2	3	4	5	6	7	1	2	3	4	5	6	7
6.	Je me sens à l'aise avec ce système	1	2	3	4	5	6	7	1	2	3	4	5	6	7
7.	J'ai eu de la facilité à apprendre comment utiliser ce système	1	2	3	4	5	6	7	1	2	3	4	5	6	7

---

Appendix

---

8. Je crois être devenu(e) rapidement efficace en utilisant ce système	1	2	3	4	5	6	7	1	2	3	4	5	6	7
9. Ce système possède toutes les fonctions et le potentiel correspondant à mes attentes	1	2	3	4	5	6	7	1	2	3	4	5	6	7
10. En général, je suis satisfait(e) de ce système	1	2	3	4	5	6	7	1	2	3	4	5	6	7

**Questions générale :**

Qu'avez-vous apprécié *le plus* : .....

.....

.....

Qu'avez-vous apprécié *le moins* : .....

.....

.....

**Merci d'avoir participé à ce test**



## ***CD Content :***

All tests, tests results, deliveries and source code files are available on this CD.

### **Hierachy :**

